

Natural

Statements-Handbuch

Bestellnummer: NAT316D031ALL

Dieses Handbuch gilt für Natural ab Version 3.1.6 für Großrechner, Version 5.1.1 für Windows und Version 5.1.1 für UNIX und OpenVMS.

Hierin enthaltene Beschreibungen unterliegen Änderungen und Ergänzungen, die in nachfolgenden Update-Serien oder Neuausgaben bekanntgegeben werden.

Anmerkungen und Verbesserungsvorschläge der Leserinnen und Leser sind sehr willkommen. Bitte richten Sie Ihre Anmerkungen an:

Software AG
Dokumentation
Uhlandstraße 12
64297 Darmstadt

Telefax: 06151-92-1612

© Juni 2002, Software AG

Alle Rechte vorbehalten

Printed in the Federal Republic of Germany

Software AG und/oder Software AG Produkte sind entweder Warenzeichen oder eingetragene Warenzeichen der Software AG. Andere hier erwähnte Produkte und Unternehmensnamen können Warenzeichen ihrer jeweiligen Eigentümer sein.

INHALTSVERZEICHNIS

VORWORT	1
Zu diesem Handbuch	1
Kapitel-Übersicht	1
Plattformspezifische Informationen	2
Beispielprogramme	2
Deutsche Rechtschreibung	2
1. SYNTAX-SYMBOLS UND OPERANDENTABELLEN	3
Operandentabellen	5
2. STATEMENTS	7
Die Statements im Überblick	7
ACCEPT/REJECT	16
ADD	20
ASSIGN	23
AT BREAK	24
AT END OF DATA	33
AT END OF PAGE	37
AT START OF DATA	43
AT TOP OF PAGE	47
BACKOUT TRANSACTION	51
BEFORE BREAK PROCESSING	55
CALL	58
CALL FILE	84
CALL LOOP	87
CALLNAT	89
CLOSE CONVERSATION	98
CLOSE DIALOG	100

Natural Statements Handbuch

CLOSE PC	101
CLOSE PRINTER	101
CLOSE WORK FILE	103
COMPOSE	105
COMPRESS	130
COMPUTE	137
CREATE OBJECT	143
DECIDE FOR	145
DECIDE ON	149
DEFINE CLASS	153
DEFINE DATA	156
DEFINE PRINTER	188
DEFINE SUBROUTINE	208
DEFINE WINDOW	214
DEFINE WORK FILE	225
DELETE	239
DISPLAY	243
DIVIDE	261
DO/DOEND	265
DOWNLOAD	267
EJECT	268
END	272
END TRANSACTION	273
ESCAPE	278
EXAMINE	282
EXAMINE TRANSLATE	290
EXPAND	293
FETCH	294
FIND	299
FOR	337
FORMAT	340

Inhaltsverzeichnis

GET	344
GET SAME	348
GET TRANSACTION DATA	352
HISTOGRAM	355
IF	362
IF SELECTION	366
IGNORE	369
INCLUDE	370
INPUT	374
INTERFACE	399
LIMIT	409
LOOP	412
METHOD	414
MOVE	419
MOVE ALL	432
MULTIPLY	435
NEWPAGE	438
NOTITLE...	443
OBTAIN	444
ON ERROR	445
OPEN CONVERSATION	448
OPEN DIALOG	449
OPTIONS	453
PASSW	454
PERFORM	457
PERFORM BREAK PROCESSING	466
PRINT	468
PROCESS	477
PROCESS COMMAND	479
PROCESS GUI	501
PROCESS REPORTER	503

Natural Statements Handbuch

PROPERTY	510
READ	511
READ WORK FILE	526
REDEFINE	533
REDUCE	536
REINPUT	537
REJECT	548
RELEASE	548
REPEAT	551
REQUEST DOCUMENT	555
RESET	568
RETRY	571
RUN	573
SEND EVENT	576
SEND METHOD	579
SEPARATE	585
SET CONTROL	594
SET GLOBALS	595
SET KEY	596
SET TIME	609
SET WINDOW	610
SKIP	611
SORT	613
STACK	621
STOP	626
STORE	628
SUBTRACT	635
SUSPEND IDENTICAL SUPPRESS	638
TERMINATE	641
UPDATE	643
UPLOAD	648

WRITE	649
WRITE TITLE	661
WRITE TRAILER	665
WRITE WORK FILE	670
3. SQL-STATEMENTS	675
Alte Statements	675
SQL-Statements-Übersicht	676
Common Set und Extended Set	678
Grundlegende Syntaxbestandteile	679
Das Natural-View-Konzept	690
Skalar-Ausdrücke (scalar-expressions)	692
Suchbedingungen (search-conditions)	698
Selektionsausdrücke (select-expressions)	705
Flexible SQL	712
Statements	716
CALLDBPROC	716
DELETE	722
INSERT	724
PROCESS SQL	727
READ RESULT SET	731
ROLLBACK	733
SELECT	735
UPDATE	749
INDEX	753

VORWORT

Zu diesem Handbuch

Dieses Handbuch gilt für alle Plattformen, auf denen Natural verwendet werden kann.

Es beschreibt die Statements, aus denen die Natural-Programmiersprache besteht.

Kapitel-Übersicht

Kapitel 1 enthält:

- Informationen zu den im Kapitel verwendeten Syntax-Symbolen, Operandentabellen und Beispielprogrammen

Kapitel 2 umfaßt folgendes:

- einen nach Funktionen gegliederten Überblick über die zur Verfügung stehenden Natural-Statements
- eine ausführliche *Beschreibung jedes einzelnen Statements*.

Kapitel 3 beschreibt die *Natural-SQL-Statements*. Mit diesen Statements können Sie in Natural-Programmen SQL direkt benutzen.

Plattformspezifische Informationen

Wo erforderlich, werden plattformspezifische Informationen in der aktuellen Dokumentation durch die folgenden Begriffe identifiziert:

Großrechner	Bezieht sich auf die Betriebssysteme OS/390, VSE/ESA, VM/CMS und BS2000/OSD sowie auf alle von Natural unter diesen Betriebssystemen unterstützten TP-Monitore.
OpenVMS	Bezieht sich auf das Betriebssystem OpenVMS.
UNIX	Bezieht sich auf alle von Natural unterstützten UNIX-Systeme.
Windows	<p>In der aktuellen Natural-Dokumentation bezieht sich “Windows” als Begriff auf die folgenden Betriebssysteme:</p> <p>In einer Natural-Entwicklungsumgebung:</p> <ul style="list-style-type: none">• Microsoft Windows NT• Microsoft Windows 2000 <p>In einer Natural-Laufzeitumgebung:</p> <ul style="list-style-type: none">• Microsoft Windows 98• Microsoft Windows NT• Microsoft Windows 2000
SQL-Datenbanken	Bezieht sich auf alle von Natural unterstützten SQL-Datenbanksysteme.

Beispielprogramme

Grundsätzlich sind die in diesem Handbuch gezeigten Beispielprogramme im *Structured Mode* geschrieben. Bei Statements, bei denen die Syntax im *Reporting Mode* stark von der Syntax im *Structured Mode* abweicht, finden Sie außerdem Verweise auf äquivalente *Reporting Mode*-Beispiele.

Die im folgenden abgebildeten Beispielprogramme sind auch online verfügbar, und zwar in der Natural-Library “SYSEXRM”.

Deutsche Rechtschreibung

Dieses Handbuch ist in der alten deutschen Rechtschreibweise abgefaßt.

SYNTAX-SYMBOLS UND OPERANDENTABELLEN

In den Diagrammen, die die Syntax der Natural-Statements beschreiben, werden folgende Symbole verwendet:

ABCDEF	Elemente, die in fettgedruckten Großbuchstaben dargestellt sind, sind Natural-Schlüsselwörter bzw. reservierte Wörter, die genauso eingegeben werden müssen wie angegeben.
<u>ABCDEF</u>	Ist von mehreren wahlweise verwendbaren Elementen, die in fettgedruckten Großbuchstaben dargestellt sind, eins unterstrichen (kein Hyperlink!), handelt es sich um das jeweils gültige Standardelement. Lassen Sie das Element weg, gilt der unterstrichene Wert.
<u>ABCDEF</u>	Ist ein Teil eines Wortes in fettgedruckten Großbuchstaben unterstrichen (kein Hyperlink!), kann der unterstrichene Teil als Abkürzung für das jeweilige Wort verwendet werden.
<i>abcdef</i>	Elemente, die <i>in Kleinbuchstaben und kursiv</i> dargestellt sind, sind variable Informationen, an deren Stelle Sie die gewünschten Angaben machen.
[]	Mehrere Elemente, die in eckigen Klammern untereinander stehen, müssen nicht unbedingt angegeben werden, sie können gegebenenfalls weggelassen werden, oder Sie können dann mindestens eine der Alternativen auswählen.
{ }	Von mehreren Elementen, die in einer geschweiften Klammer untereinander stehen, muß eines angegeben werden.
	Eines der durch diesen senkrechten Strich voneinander getrennten Elemente muß eingegeben werden.
...	Drei Punkte nach einem Element bedeuten, daß das Element mehrmals angegeben werden darf. Gegebenenfalls gibt eine Zahl nach den Punkten an, wie oft das Element angegeben werden darf. Ist das Element vor den drei Punkten ein in eckige oder geschweifte Klammern eingeschlossener Ausdruck, gilt die Ellipse für den gesamten in Klammern stehenden Ausdruck.
,...	Ein Komma und drei Punkte nach einem Element bedeuten, daß das Element mehrmals angegeben werden darf, wobei die einzelnen Angaben durch Kommas voneinander getrennt werden müssen. Gegebenenfalls gibt eine Zahl nach dem Komma und den drei Punkten an, wie oft das Element angegeben werden darf. Ist das Element vor dem Komma und den drei Punkten ein in eckige oder geschweifte Klammern eingeschlossener Ausdruck, gilt das Komma und die Ellipse für den gesamten in Klammern stehenden Ausdruck.

:....	<p>Ein Doppelpunkt und drei Punkte nach einem Element bedeuten, daß das Element mehrmals angegeben werden darf, wobei die einzelnen Angaben durch Doppelpunkte voneinander getrennt werden müssen. Gegebenenfalls gibt eine Zahl nach dem Doppelpunkt und den drei Punkten an, wie oft das Element angegeben werden darf.</p> <p>Ist das Element vor dem Doppelpunkt und den drei Punkten ein in eckige oder geschweifte Klammern eingeschlossener Ausdruck, gilt der Doppelpunkt und die Ellipse für den gesamten in Klammern stehenden Ausdruck.</p>
Andere Symbole	<p>Alle anderen Symbole außer den in dieser Tabelle definierten müssen genauso eingegeben werden wie angegeben.</p> <p><i>Ausnahme:</i> Der skalare SQL-Verkettungsoperator wird durch zwei senkrechte Striche dargestellt, die genauso eingegeben werden müssen, wie sie in der Syntax-Definition erscheinen.</p>

Beispiel:

$$\text{WRITE [USING] } \left\{ \begin{array}{l} \text{FORM} \\ \text{MAP} \end{array} \right\} \text{ operand1 [operand2...]}$$

- **WRITE**, **USING**, **MAP** und **FORM** sind Natural-Schlüsselwörter, die Sie genauso eingeben müssen wie angegeben.
- *operand1* und *operand2* sind Variablen, an deren Stelle Sie die Namen der betreffenden Objekte eingeben.
- Die geschweiften Klammern bedeuten, daß Sie entweder **FORM** oder **MAP** angeben können, aber eins von beiden angeben müssen.
- Die eckigen Klammern bedeuten, daß **USING** und *operand2* optionale Elemente sind, die Sie angeben können, aber nicht müssen.
- Die drei Punkte bedeuten, daß Sie *operand2* mehrmals angeben können.

Operandentabellen

Enthält die Syntax eines Natural-Statements einen oder mehrere *Operanden*, so finden Sie bestimmte Informationen zu diesen Operanden jeweils in der folgenden Tabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S A G N/M	A N P I F B D T L C G O	ja/nein	ja/nein

Die Tabelle enthält folgende Informationen zu jedem Operanden:

Mögliche Struktur

Gibt an, welche Struktur der Operand haben darf:

C	Konstante.
S	Skalar (ein Feld bzw. eine Variable mit nur einer Ausprägung; d.h. weder ein Array-Bereich noch eine Gruppe).
A	Array.
G	Gruppe.
N/M	Natural-Systemvariable: N = Es dürfen alle Systemvariablen verwendet werden. M = Nur modifizierbare Systemvariablen dürfen verwendet werden. Welche Systemvariablen modifizierbar sind, steht im Kapitel Systemvariablen im <i>Natural-Referenzhandbuch</i> .

Mögliche Formate

Gibt an, welche Formate der Operand haben darf:

A	Alphanumerisch
N	Numerisch ungepackt
P	Gepackt numerisch
I	Integer (= ganzzahlig)
F	Floating point (= Gleitkomma)
B	Binär
D	Datum
T	Time (= Zeit)
L	Logisch
C	Attributkontrolle
G	GUI-Handle
O	Objekt-Handle

Referenzierung erlaubt

Gibt an, ob der Operand über ein Statement-Label bzw. die Sourcecode-Zeilenummer referenziert werden darf.

Dynam. Definition

Gibt an, ob der Operand dynamisch im Programm definiert werden darf. Dies ist nur im *Reporting Mode* möglich.

STATEMENTS

Statements sind die “Wörter”, aus denen die Natural-Programmiersprache besteht.

Dieses Kapitel enthält:

- einen nach Funktionen gegliederten *Überblick* über die zur Verfügung stehenden Natural-Statements
- eine ausführliche *Beschreibung jedes einzelnen Statements*.

Die Statements im Überblick

Der vorliegende Abschnitt bietet einen Überblick über die zur Verfügung stehenden Natural-Statements. Die Statements lassen sich bezüglich ihrer Funktionen in folgende Gruppen aufteilen:

- Datenbankzugriffe und Datenbankänderungen
- Arithmetische Funktionen und Datenzuweisungen
- Ausführung von Verarbeitungsschleifen
- Erstellung von Ausgabe-Reports
- Bildschirmgenerierung für interaktive Verarbeitung
- Verarbeitung logischer Bedingungen
- Aufrufen von Programmen und Unterprogrammen
- Verarbeitung von Arbeitsdateien
- Komponentenbasierte Programmierung
- Ereignisgesteuerte Programmierung
- Sonstige Statements.

Datenbankzugriffe und Datenbankänderungen

Die folgenden Statements greifen auf in einer Datenbank gespeicherte Daten zu:

READ	Lesen einer Datei in physischer oder logischer Reihenfolge der Datensätze.
FIND	Auswählen von Datensätzen aufgrund bestimmter Kriterien.
HISTOGRAM	Lesen von Werten eines Datenbankfeldes.
GET	Lesen eines Datensatzes mit einer bestimmten ISN (Interne Satznummer) bzw. SNR (Satznummer).
GET SAME	Erneutes Lesen des gerade verarbeiteten Datensatzes.
ACCEPT/REJECT	Annehmen/Ablehnen von Datensätzen aufgrund bestimmter Kriterien.
PASSW	Angabe eines Paßworts zur Zugriffsberechtigung auf eine paßwort-geschützte Datei.
LIMIT	Begrenzen der Anzahl der Ausführungen einer READ-, FIND- oder HISTOGRAM-Schleife.
STORE	Anlegen eines neuen Datensatzes in der Datenbank.
UPDATE	Ändern eines Datensatzes in der Datenbank.
DELETE	Löschen eines Datensatzes von der Datenbank.
END TRANSACTION	Festlegen des Endes einer logischen Transaktion.
BACKOUT TRANSACTION	Abbrechen einer nicht vollständig abgeschlossenen logischen Transaktion.
GET TRANSACTION DATA	Lesen von Transaktionsdaten, die mit einem vorhergegangenen END TRANSACTION-Statement gespeichert wurden.
RETRY	Erneuter Versuch, einen Datensatz zu lesen, der vorher von einem anderen Benutzer benutzt wurde.
AT START OF DATA	Ausführen von Statements, wenn in einer Schleife der erste Datensatz verarbeitet wird.
AT END OF DATA	Ausführen von Statements, nachdem in einer Schleife der letzte Datensatz verarbeitet wurde.
AT BREAK	Ausführen von Statements bei einem Wertwechsel in einem bestimmten Feld (Gruppenwechsel).
BEFORE BREAK PROCESSING	Ausführen von Statements vor einer Gruppenwechsel-Verarbeitung.
PERFORM BREAK PROCESSING	Sofortiges Ausführen einer Gruppenwechsel-Verarbeitung.

Arithmetische Funktionen und Datenzuweisungen

Die folgenden Statements werden verwendet, um arithmetische Operationen sowie Datenzuweisungen durchzuführen:

COMPUTE	Rechenoperationen ausführen oder Feldern Werte zuweisen.
ADD	Addieren von Operanden.
SUBTRACT	Subtrahieren von Operanden.
MULTIPLY	Multiplizieren von Operanden.
DIVIDE	Dividieren eines Operanden durch einen anderen.
MOVE	Übertragen eines Operandenwertes in ein oder mehrere Feld/er.
MOVE ALL	Übertragen sämtlicher Werte einer bestimmten Größe in ein anderes Feld.
COMPRESS	Aneinanderreihen mehrerer Feldwerte in einem Feld.
SEPARATE	Aufteilen eines Feldwertes in zwei oder mehr Felder.
EXAMINE	Absuchen eines Feldes nach einem bestimmten Wert und anschließend Ersetzen des Wertes und/oder Zählen, wie oft der Wert vorkommt.
RESET	Zurücksetzen eines Feldwertes auf Null (numerisches Feld) bzw. Leerwert (alphanumerisches Feld) oder auf einen Ausgangswert.

Schleifenverarbeitung

Die folgenden Statements werden in Verbindung mit der Ausführung von Verarbeitungsschleifen verwendet:

REPEAT	Initiieren einer Verarbeitungsschleife (und Beenden in Abhängigkeit von einer bestimmten Bedingung).
FOR	Initiieren einer Verarbeitungsschleife und Steuerung der Anzahl der Schleifendurchläufe.
ESCAPE	Ausführung einer Verarbeitungsschleife abbrechen.

Erstellen von Ausgabe-Reports

Die folgenden Statements werden bei der Erzeugung von Ausgabe-Reports verwendet:

FORMAT	Spezifizieren von Ausgabe-Parametern.
DISPLAY	Ausgabe von Feldwerten in Spalten untereinander.
WRITE/PRINT	Ausgabe von Feldwerten ohne Spalteneinteilung.
WRITE TITLE	Überschreiben einer Standard-Seitenüberschrift mit einer eigenen Seitenüberschrift.
WRITE TRAILER	Ausgabe eines Fußzeilentextes, der auf jeder Ausgabeseite erscheinen soll.
AT TOP OF PAGE	Spezifizieren der Verarbeitung, die beim Beginn einer neuen Ausgabeseite ausgeführt werden soll.
AT END OF PAGE	Spezifizieren der Verarbeitung, die beim Erreichen des Endes einer Ausgabeseite ausgeführt werden soll.
SKIP	Generieren von Leerzeilen in der Ausgabe.
EJECT	Seitenvorschub ohne Titel und Überschriften.
NEWPAGE	Seitenvorschub mit Titel und Überschriften.
SUSPEND IDENTICAL SUPPRESS	Aussetzen der "Identical Suppress"-Bedingung für einen einzelnen Datensatz.
DEFINE PRINTER	Bestimmen des Druckers oder logischen Zielorts, an dem ein Report ausgegeben werden soll.
CLOSE PRINTER	Schließen eines Druckers.

Bildschirmgenerierung für interaktive Verarbeitung

Die folgenden Statements werden in Verbindung mit der Verwendung von Bildschirmmasken (Maps) bei interaktiver Datenverarbeitung benutzt:

INPUT	Erstellen einer formatierten Map zur Datenausgabe/-eingabe.
REINPUT	Erneutes Ausführen eines INPUT-Statements (falls die auf das INPUT-Statement erfolgte Dateneingabe fehlerhaft war).
DEFINE WINDOW	Größe, Position und Attribute eines Windows festlegen.
SET WINDOW	Aktivieren und Deaktivieren eines Windows.

Verarbeitung logischer Bedingungen

Mit den folgenden Statements wird die Ausführung von Statements in Abhängigkeit von Bedingungen gesteuert, die während der Ausführung eines Natural-Programms auftreten:

IF	Ausführen von Statements aufgrund einer logischen Bedingung.
IF SELECTION	Prüfen, ob in einer Reihe von alphanumerischen Feldern genau ein Wert enthält.
DECIDE FOR	Ausführen von Statements aufgrund von logischen Bedingungen.
DECIDE ON	Ausführen von Statements aufgrund des Inhaltes einer Variablen.
ON ERROR	Abfangen von Laufzeitfehlern, die normalerweise eine Fehlermeldung und den Abbruch des ausgeführten Programms bewirken würden.

Aufrufen von Programmen und Unterprogrammen

Die folgenden Statements werden zum Aufrufen von Programmen und Unterprogrammen verwendet:

FETCH	Aufrufen eines Natural-Programms.
CALLNAT	Aufrufen eines Natural-Subprogramms.
PERFORM	Aufrufen einer Natural-Subroutine.
DEFINE SUBROUTINE	Definieren einer Natural-Subroutine.
ESCAPE	Abbrechen eines Unterprogramms.
CALL	Aufrufen eines Nicht-Natural-Programms von einem Natural-Programm aus.
CALL FILE	Aufrufen eines Nicht-Natural-Programms, um einen Datensatz von einer Nicht-Adabas-Datei zu lesen.
CALL LOOP	Generierung einer Verarbeitungsschleife, die den Aufruf eines Nicht-Natural-Programms beinhaltet.

Arbeitsdatei-Verarbeitung

Die folgenden Statements werden verwendet, um Daten auf eine physisch-sequentielle (nicht-Adabas) Arbeitsdatei zu schreiben bzw. von dieser zu lesen:

WRITE WORK FILE	Schreibt Daten auf eine Arbeitsdatei.
READ WORK FILE	Liest Daten von einer Arbeitsdatei.
CLOSE WORK FILE	Schließt eine Arbeitsdatei.
DEFINE WORK FILE	Weist einer Arbeitsdatei einen Dateinamen zu.

Komponentenbasierte Programmierung

Die folgenden Statements werden zur komponentenbasierten Programmierung verwendet:

DEFINE CLASS	Gibt innerhalb eines Natural-Klassenmoduls eine Klasse an.
CREATE OBJECT	Erstellt ein Objekt (auch bekannt als "Instanz") einer gegebenen Klasse.
SEND METHOD	Ruft eine Methode eines Objekts auf.
INTERFACE	Definiert eine Schnittstelle (eine Sammlung von Methoden und Eigenschaften) für eine bestimmte Funktion einer Klasse.
METHOD	Weist außerhalb einer Schnittstellendefinition ein Unterprogramm als Implementierung einer Methode zu.
PROPERTY	Weist außerhalb einer Schnittstellendefinition eine Objektdaten-Variable als Implementierung einer Eigenschaft zu.

Ereignisgesteuerte Programmierung

Die folgenden Statements werden zur ereignisgesteuerten Programmierung verwendet:

OPEN DIALOG	Öffnet einen Dialog.
CLOSE DIALOG	Schließt einen Dialog.
SEND EVENT	Löst einen benutzerdefinierten Event aus.
PROCESS GUI	Führt eine Standardprozedur in einer ereignisgesteuerten Anwendung aus.

Sonstige Statements

DEFINE DATA	Definiert die Datenelemente, die in einem Natural-Programm oder -Unterprogramm verwendet werden sollen.
END	Zeigt das Ende des Sourcecodes eines Natural-Programms bzw. -Unterprogramms an.
EXPAND	Erweitert den dynamischen Variablen zugewiesenen Speicher auf eine angegebene Größe.
EXAMINE TRANSLATE	Umsetzen der Zeichen eines Feldes in Groß- oder Kleinschreibung oder in andere Zeichen.
INCLUDE	Einfügen von Natural-Copycode während der Kompilierung.
PROCESS COMMAND	Aufrufen eines Kommando-Prozessors.
REDUCE	Verringert den dynamischen Variablen zugewiesenen Speicher.
RELEASE	Löschen aller im Natural-Stack gehaltenen Daten; Freigabe aller Daten, die über eine RETAIN-Klausel in einem FIND-Statement gehalten wurden; Zurücksetzen von globalen Variablen auf die ursprünglichen Werte.
REQUEST DOCUMENT	Ermöglicht den Zugriff auf ein externes System.
RUN	Kompilieren und Ausführen eines Source-Programms.
SET CONTROL	Ausführen eines Natural-Terminalkommandos aus einem Natural-Programm heraus.
SET KEY	Zuweisen von Funktionen zu Funktionstasten.
SETTIME	Setzen eines zeitlichen Bezugspunkts für eine *TIMD-Systemvariable.
SORT	Sortieren von Datensätzen unter Verwendung des Sortierprogramms des Betriebssystems.
STACK	Zwischenlagern von Daten bzw. Kommandos im Natural-Stack.
STOP	Abbrechen der Ausführung einer Anwendung.
TERMINATE	Abbrechen der Natural-Session.

ACCEPT/REJECT

$$\left\{ \begin{array}{l} \text{ACCEPT} \\ \text{REJECT} \end{array} \right\} [\text{IF}] \textit{logical-condition}$$

Funktion

Mit den Statements ACCEPT und REJECT können Sie eine logische Bedingung (*logical-condition*) angeben, aufgrund der ein gelesener Datensatz akzeptiert (ACCEPT) oder zurückgewiesen (REJECT) werden soll.

Beide Statements können in Verbindung mit Statements eingesetzt werden, die Datensätze in einer Verarbeitungsschleife lesen (FIND, READ, HISTOGRAM, CALL FILE, SORT oder READ WORK FILE). Die logische Bedingung wird erst ausgewertet, *nachdem* ein Datensatz gelesen worden ist.

Wenn ein ACCEPT- bzw. REJECT-Statement ausgeführt wird, bezieht es sich auf die innerste gerade aktive Verarbeitungsschleife, die mit einem der oben genannten Statements initiiert wurde.

Befindet sich ein ACCEPT- oder REJECT-Statement in einer Subroutine und wird aufgrund der logischen Bedingung ein Datensatz zurückgewiesen, so wird die Subroutine automatisch beendet und die Verarbeitung mit dem nächsten Datensatz der innersten gerade aktiven Verarbeitungsschleife fortgesetzt.

In der logischen Bedingung benutzte Felder

Bei der Angabe einer logischen Bedingung können Sie Datenbankfelder oder Benutzervariablen benutzen. Weitere Informationen zu logischen Bedingungen finden Sie im *Natural-Referenzhandbuch*.

Verwenden Sie ACCEPT/REJECT in Verbindung mit einem HISTOGRAM-Statement, so darf die logische Bedingung nur in Abhängigkeit von dem im HISTOGRAM-Statement benutzten Datenbankfeld spezifiziert werden.

Die Verarbeitung mehrerer ACCEPT/REJECT-Statements

Pro Verarbeitungsschleife genügt in der Regel ein ACCEPT- bzw. REJECT-Statement. Wollen Sie in einer Verarbeitungsschleife mehrere ACCEPT/REJECT-Statements *unmittelbar hintereinander* verwenden, so beachten Sie bitte folgende Regeln:

- Befinden sich innerhalb einer Verarbeitungsschleife mehrere ACCEPT/REJECT-Statements direkt hintereinander, so werden sie in der angegebenen Reihenfolge verarbeitet.
- Wird aufgrund einer erfüllten ACCEPT-Bedingung ein Datensatz akzeptiert, so werden die unmittelbar nachfolgenden ACCEPT/REJECT-Statements ignoriert.
- Wird aufgrund einer erfüllten REJECT-Bedingung ein Datensatz zurückgewiesen, so werden die unmittelbar nachfolgenden ACCEPT/REJECT-Statements ignoriert.
- Geht die Verarbeitung bis zum letzten ACCEPT/REJECT-Statement, so entscheidet dieses letzte Statement, ob der betreffende Datensatz akzeptiert wird oder nicht.

Befindet sich zwischen zwei ACCEPT/REJECT-Statements ein anderes Statement, so werden beide ACCEPT/REJECT-Statements unabhängig voneinander verarbeitet.

Begrenzte Anzahl der Schleifendurchläufe

Ist die Anzahl der Durchläufe einer Verarbeitungsschleife durch ein LIMIT-Statement oder eine andere Einschränkung begrenzt, so gilt diese für die Anzahl der gelesenen Datensätze, und zwar unabhängig davon, wieviele der gelesenen Datensätze aufgrund eines ACCEPT- oder REJECT-Statements akzeptiert oder zurückgewiesen werden.

Hold-Status

Eine ACCEPT/REJECT-Verarbeitung hat keinen Einfluß darauf, ob ein im “Hold” gehaltener Datensatz freigegeben wird — es sei denn, der Profilparameter RI ist auf RI=ON gesetzt (dieser Parameter ist nur auf Großrechnern verfügbar; vgl. *Natural-Referenzhandbuch* und *Natural Operations for Mainframes Documentation*).

Beispiel 1

```

/* EXAMPLE 'ACREX1S': ACCEPT (STRUCTURED MODE)
DEFINE DATA LOCAL
  1 EMPLOY-VIEW VIEW OF EMPLOYEES
    2 NAME
    2 SEX
    2 MAR-STAT
END-DEFINE
LIMIT 50
READ EMPLOY-VIEW
  ACCEPT IF SEX='M' AND MAR-STAT = 'S'
  WRITE NOTITLE '=' NAME '=' SEX 5X '=' MAR-STAT
END-READ
END

```

NAME: MORENO	S E X: M	MARITAL STATUS: S
NAME: VAUZELLE	S E X: M	MARITAL STATUS: S
NAME: BAILLET	S E X: M	MARITAL STATUS: S
NAME: HEURTEBISE	S E X: M	MARITAL STATUS: S
NAME: LION	S E X: M	MARITAL STATUS: S
NAME: DEZELUS	S E X: M	MARITAL STATUS: S
NAME: BOYER	S E X: M	MARITAL STATUS: S
NAME: BROUSSE	S E X: M	MARITAL STATUS: S
NAME: DROMARD	S E X: M	MARITAL STATUS: S
NAME: DUC	S E X: M	MARITAL STATUS: S
NAME: BEGUERIE	S E X: M	MARITAL STATUS: S
NAME: FOREST	S E X: M	MARITAL STATUS: S
NAME: GEORGES	S E X: M	MARITAL STATUS: S
NAME: BOUCLY	S E X: M	MARITAL STATUS: S

Äquivalentes Reporting-Mode-Beispiel: siehe Programm ACREX1R in Library SYSEXRM.

Beispiel 2

```

/* EXAMPLE 'ACREX2S': ACCEPT/REJECT (STRUCTURED MODE)
DEFINE DATA LOCAL
  1 EMPLOY-VIEW VIEW OF EMPLOYEES
    2 NAME
    2 FIRST-NAME
    2 SALARY (1)
  1 #PROC-COUNT (N8) INIT <0>
END-DEFINE
EMP. FIND EMPLOY-VIEW WITH NAME = 'JACKSON'
  WRITE NOTITLE *COUNTER NAME FIRST-NAME 'SALARY:' SALARY(1)
/* *****
  ACCEPT IF SALARY (1) LT 50000
  WRITE *COUNTER 'ACCEPTED FOR FURTHER PROCESSING'
/* *****
  REJECT IF SALARY (1) GT 30000
  WRITE *COUNTER 'NOT REJECTED'
/* *****
  ADD 1 TO #PROC-COUNT
END-FIND
SKIP 2
WRITE NOTITLE 'TOTAL PERSONS FOUND ' *NUMBER (EMP.)
  / 'TOTAL PERSONS SELECTED' #PROC-COUNT
END

```

1	JACKSON	CLAUDE	SALARY:	33000
1	ACCEPTED FOR FURTHER	PROCESSING		
2	JACKSON	FORTUNA	SALARY:	36000
2	ACCEPTED FOR FURTHER	PROCESSING		
3	JACKSON	CHARLIE	SALARY:	23000
3	ACCEPTED FOR FURTHER	PROCESSING		
3	NOT REJECTED			
TOTAL PERSONS FOUND				3
TOTAL PERSONS SELECTED				1

Äquivalentes Reporting-Mode-Beispiel: siehe Programm ACREX2R in Library SYSEXRM.

ADD

ADD [ROUNDED] operand1... TO operand2

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S A N	N P I F D T	ja	nein
Operand2	S A M	N P I F D T	ja	ja

ADD [ROUNDED] operand1... GIVING operand2

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S A N	N P I F D T	ja	nein
Operand2	S A M	A N P I F B D T	ja	ja

Verwandtes Statement

COMPUTE.

Funktion

Mit dem ADD-Statement können Sie zwei oder mehrere Operanden addieren.

Operanden

Zu dem Zeitpunkt, zu dem das ADD-Statement ausgeführt wird, muß jeder der Operanden einen gültigen Wert enthalten.

Zu Additionen mit Arrays siehe auch den Abschnitt **Arithmetische Operationen mit Arrays** im des *Natural-Referenzhandbuch*.

Zum Format der Operanden siehe auch den Abschnitt **Formatwahl im Hinblick auf die Verarbeitungszeit** im *Natural-Referenzhandbuch*.

Ergebnisfeld (*operand2*)

TO

Verwenden Sie das Schlüsselwort TO, so wird *operand2* bei der Addition mit hinzuaddiert und enthält anschließend die Summe.

GIVING

Verwenden Sie das Schlüsselwort GIVING, so ist *operand2* lediglich das Ergebnisfeld. Ist *operand2* hierbei alphanumerisch definiert, wird das Ergebnis in alphanumerisches Format umgesetzt.

Verwenden Sie ein Datenbankfeld als Ergebnisfeld, ändert sich allenfalls programmintern der Wert des Feldes; eine Änderung des Feldwertes auf der Datenbank ergibt sich hieraus nicht.

ROUNDED

Wünschen Sie das Ergebnis gerundet, geben Sie das Schlüsselwort ROUNDED an. Die für das Runden gültigen Regeln finden Sie unter **Arithmetische Operationen** im *Natural-Referenzhandbuch*.

Beispiel

```

* EXAMPLE 'ADDEX1': ADD
*****
DEFINE DATA LOCAL
  1 #A (P2)
  1 #B (P1.1)
  1 #C (P1)
  1 #DATE (D)
  1 #ARRAY1 (P5/1:4,1:4) INIT (2,*) <5>
  1 #ARRAY2 (P5/1:4,1:4) INIT (4,*) <10>
END-DEFINE
*
ADD +5 -2 -1 GIVING #A
WRITE NOTITLE 'ADD +5 -2 -1 GIVING #A' 15X '=' #A
*
ADD .231 3.6 GIVING #B
WRITE /      'ADD .231 3.6 GIVING #B' 15X '=' #B
*
ADD ROUNDED 2.9 3.8 GIVING #C
WRITE /      'ADD ROUNDED 2.9 3.8 GIVING #C' 8X '=' #C
*
MOVE *DATX TO #DATE
ADD 7 TO #DATE
WRITE / 'CURRENT DATE:'      *DATX (DF=L) 13X
      'CURRENT DATE + 7:' #DATE (DF=L)
*
WRITE /      '#ARRAY1 AND #ARRAY2 BEFORE ADDITION'
      /      '=' #ARRAY1 (2,*)
      /      '=' #ARRAY2 (4,*)
ADD #ARRAY1 (2,*) TO #ARRAY2 (4,*)
WRITE /      '#ARRAY1 AND #ARRAY2 AFTER ADDITION'
      /      '=' #ARRAY1 (2,*)
      /      '=' #ARRAY2 (4,*)
*
END

```

ADD +5 -2 -1 GIVING #A	#A:	2							
ADD .231 3.6 GIVING #B	#B:	3.8							
ADD ROUNDED 2.9 3.8 GIVING #C	#C:	7							
CURRENT DATE: 1999-01-19	CURRENT DATE + 7:	1999-01-26							
#ARRAY1 AND #ARRAY2 BEFORE ADDITION									
#ARRAY1:	5	5	5	5	#ARRAY2:	10	10	10	10
#ARRAY1 AND #ARRAY2 AFTER ADDITION									
#ARRAY1:	5	5	5	5	#ARRAY2:	15	15	15	15

ASSIGN

Siehe COMPUTE-Statement.

AT BREAK

Structured-Mode-Syntax

```
[AT] BREAK [(r)] [OF] operand1 [/n/]
      statement...
END-BREAK
```

Reporting-Mode-Syntax

```
[AT] BREAK [(r)] [OF] operand1 [/n/]
      {
        statement
      DO statement... DOEND }
```

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S	A N P I F B D T L	ja	nein

Anmerkung:

Für Operand1 sind keine dynamischen oder großen Variablen zulässig.

Verwandte Statements

BEFORE BREAK PROCESSING, FIND, READ, HISTOGRAM, SORT, READ WORK FILE.

Funktion

Das Statement AT BREAK dient dazu, in einer mit FIND, READ, HISTOGRAM, SORT oder READ WORK FILE initiierten Verarbeitungsschleife eine an einen automatischen Gruppenwechsel geknüpfte Verarbeitung anzugeben. Mit dem AT BREAK-Statement können Sie ein oder mehrere andere Statements angeben, die jedesmal ausgeführt werden sollen, wenn der Wert eines bestimmten Feldes sich ändert.

Ein AT BREAK-Statement-Block wird nur ausgeführt, wenn das Objekt, das den Statement-Block enthält, zu dem Zeitpunkt, zu dem die Gruppenwechsel-Bedingung auftritt, aktiv ist.

Es ist auch möglich, mit einer AT BREAK-Verarbeitung eine weitere Verarbeitungsschleife zu initiieren. Die Schleife muß allerdings innerhalb der AT BREAK-Verarbeitung wieder beendet werden.

Dieses Statement ist non-prozedural (das heißt, seine Ausführung hängt von einem Ereignis ab, nicht davon, wo im Programm es steht).

Referenzierungsnotation (*r*)

Ein AT BREAK-Statement wird zum letztenmal ausgeführt, wenn eine mit FIND, READ, HISTOGRAM, SORT oder READ WORK FILE initiierte Verarbeitungsschleife beendet wird. Normalerweise bezieht sich das AT BREAK-Statement hierbei auf die äußerste aktive Schleife.

Wollen Sie, daß sich die abschließende AT BREAK-Verarbeitung auf eine andere offene Schleife bezieht (die Schleife, in der das AT BREAK-Statement steht, oder eine äußere Schleife), so verwenden Sie hierzu die Notation “(*r*)”, wobei *r* das Statement-Label bzw. die Sourcecode-Zeilenummer des betreffenden schleifeninitiiierenden Statements ist.

Beispiel:

```
0110 ...
0120 READ ...
0130   FIND ...
0140     FIND ...
0150       AT BREAK ...
0160         FIND ...
0170           END-FIND
0180             END-BREAK
0190               END-FIND
0200                 END-FIND
0210                   END-READ
0220 ...
```

In diesem Beispiel bezieht sich die abschließende `AT BREAK`-Bedingung auf die in Zeile 0120 initiierte `READ`-Schleife. Es wäre auch möglich, sie an eine der in Zeile 0130 bzw. 0140 initiierten `FIND`-Schleifen zu knüpfen, nicht jedoch an die in Zeile 0160 initiierte.

Soll eine ganze Hierarchie von `AT BREAK`-Statements sich auf eine andere als die aktive Schleife beziehen, so müssen Sie die Notation “(*r*)” bei dem ersten `AT BREAK`-Statement angeben; sie bezieht sich dann auch auf alle innerhalb der Hierarchie folgenden `AT BREAK`-Statements.

Kontrollfeld (*operand1*)

In der Regel wird als Kontrollfeld ein Datenbankfeld verwendet. Sie können aber auch eine Benutzervariable nehmen, müssen diese allerdings vor der Gruppenwechsel-Verarbeitung definiert haben (siehe BEFORE BREAK PROCESSING-Statement).

/n/

Sie haben auch die Möglichkeit, einen *Teil eines Feldes* zum Kontrollfeld zu machen: mit der Notation “/n/” geben Sie an, daß nur die ersten *n* Stellen des Feldes als Kontrollfeld dienen sollen, d.h. das AT BREAK-Statement wird nur ausgeführt, wenn der Wert der ersten *n* Stellen sich ändert. Diese Möglichkeit besteht allerdings nur bei Feldern, die das Format A, N oder P haben.

Sie können auch eine bestimmte Ausprägung eines Arrays als Kontrollfeld verwenden.

Ein AT BREAK-Statement wird immer dann ausgeführt, wenn ein Gruppenwechsel stattfindet, das heißt, wenn der Wert des Kontrollfeldes sich ändert. Es wird außerdem ausgeführt, nachdem alle Datensätze in der Schleife, auf die sich das AT BREAK-Statement bezieht, verarbeitet worden sind.

Beispiel 1

```

/* EXAMPLE 'ATBEX1S': AT BREAK (STRUCTURED MODE)
/*****
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 COUNTRY
  2 NAME
END-DEFINE
/*****
LIMIT 10
READ EMPLOY-VIEW BY CITY
  AT BREAK OF CITY
    SKIP 1
    END-BREAK
  DISPLAY NOTITLE CITY (IS=ON) COUNTRY (IS=ON) NAME
END-READ
/*****
END

```

CITY	COUNTRY	NAME
AIKEN	USA	SENKO
AIX EN OTHE.	F	GODEFROY
AJACCIO		CANALE
ALBERTSLUND	DK	PLOUG
ALBUQUERQUE	USA	HAMMOND ROLLING FREEMAN LINCOLN
ALFRETON	UK	GOLDBERG
ALICANTE	E	GOMEZ

Äquivalentes Reporting-Mode-Beispiel: siehe Programm ATBEX1R in Library SYSEXRM.

Beispiel 2

```

/* EXAMPLE 'ATBEX2': AT BREAK USING /N/ NOTATION
/*****
DEFINE DATA LOCAL
  1 EMPLOY-VIEW VIEW OF EMPLOYEES
    2 DEPT
    2 NAME
END-DEFINE
/*****
LIMIT 10
READ EMPLOY-VIEW BY DEPT STARTING FROM 'A'
  AT BREAK OF DEPT /4/
    SKIP 1
  END-BREAK
  DISPLAY NOTITLE DEPT NAME
END-READ
/*****
END

```

DEPARTMENT CODE	NAME
ADMA01	JENSEN
ADMA01	PETERSEN
ADMA01	MORTENSEN
ADMA01	MADSEN
ADMA01	BUHL
ADMA02	HERMANSEN
ADMA02	PLOUG
ADMA02	HANSEN
COMP01	HEURTEBISE
COMP01	TANCHOU

Systemfunktionen

Natural-Systemfunktionen können in Verbindung mit einem AT BREAK-Statement verwendet werden, wie im Kapitel **Systemfunktionen** des *Natural-Referenzhandbuchs* beschrieben.

Gruppenwechsel auf mehreren Ebenen

Innerhalb einer Verarbeitungsschleife innerhalb desselben Programm-Moduls können Sie mehrere AT BREAK-Statements verwenden. Damit schaffen Sie eine Hierarchie von AT BREAK-Statements, und zwar unabhängig davon, ob die AT BREAK-Statements unmittelbar aufeinander folgen oder zwischen ihnen noch andere Statements stehen. Das erste AT BREAK-Statement befindet sich auf der untersten Ebene der Hierarchie, jedes weitere auf einer nächsthöheren.

Sie können für jede Schleife innerhalb einer Schleife eine eigene AT BREAK-Hierarchie aufbauen.

Beispiel — Structured Mode:

```
FIND ...
  AT BREAK
  ...
  END-BREAK
  AT BREAK
  ...
  END-BREAK
  AT BREAK
  ...
  END-BREAK
END-FIND
...
```

Beispiel — Reporting Mode:

```
FIND ...
  AT BREAK
  DO
  ...
  DOEND
  AT BREAK
  DO
  ...
  DOEND
...
```

Bei einem Gruppenwechsel auf einer bestimmten Ebene werden auch alle AT BREAK-Statements auf jeweils untergeordneten Ebenen der Hierarchie ausgeführt, unabhängig davon, ob im Kontrollfeld einer unteren Ebene ebenfalls ein Gruppenwechsel stattgefunden hat.

Der Übersichtlichkeit halber empfiehlt es sich, die einzelnen AT BREAK-Statements einer Hierarchie unmittelbar aufeinanderfolgen zu lassen.

Beispiel 3

```

/* EXAMPLE 'ATBEX5S': AT BREAK WITH MULTIPLE BREAK LEVELS
/*
/*          (STRUCTURED MODE)
/******
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 DEPT
  2 NAME
  2 LEAVE-DUE
1 LEAVE-DUE-L (N4)
END-DEFINE
/******
LIMIT 5
FIND EMPLOY-VIEW WITH CITY = 'PHILADELPHIA' OR = 'PITTSBURGH'
      SORTED BY CITY DEPT
MOVE LEAVE-DUE TO LEAVE-DUE-L
DISPLAY CITY (IS=ON) DEPT (IS=ON) NAME LEAVE-DUE-L
  AT BREAK OF DEPT
    WRITE NOTITLE /
      T*DEPT OLD(DEPT) T*LEAVE-DUE-L SUM(LEAVE-DUE-L) /
  END-BREAK
  AT BREAK OF CITY
    WRITE NOTITLE
      T*CITY OLD(CITY) T*LEAVE-DUE-L SUM(LEAVE-DUE-L) //
  END-BREAK
END-FIND
/******
END

```

CITY	DEPARTMENT CODE	NAME	LEAVE-DUE-L
PHILADELPHIA	MGMT30	WOLF-TERROINE	8
		MACKARNES	12
	MGMT30		20
	TECH10	BUSH	8
		NETTLEFOLDS	7
TECH10		15	
PHILADELPHIA			35
PITTSBURGH	MGMT10	FLETCHER	3
	MGMT10		3
PITTSBURGH			3

Äquivalentes Reporting-Mode-Beispiel: siehe Programm ATBEX5R in Library SYSEXRM.

AT END OF DATA

Structured-Mode-Syntax

```
[AT] END [OF] DATA [(r)]  
    statement...  
END-ENDDATA
```

Reporting-Mode-Syntax

```
[AT] END [OF] DATA [(r)]  
    { statement  
      DO statement... DOEND }
```

Verwandte Statements

AT START OF DATA, FIND, READ, HISTOGRAM, SORT, READ WORK FILE.

Funktion

Mit dem Statement AT END OF DATA können Sie eine Verarbeitung angeben, die ausgeführt werden soll, nachdem in einer Verarbeitungsschleife alle Datensätze verarbeitet worden sind. Das AT END OF DATA-Statement muß im selben Objektmodul stehen wie das Statement, mit dem die Schleife initiiert wurde.

Dieses Statement ist non-prozedural (das heißt, seine Ausführung hängt von einem Ereignis ab, nicht davon, wo im Programm es steht).

Einschränkungen

Das Statement kann nur bei einer Verarbeitungsschleife eingesetzt werden, die mit einem der folgenden Statements initiiert wurde: FIND, READ, READ WORK FILE, HISTOGRAM oder SORT. Pro Schleife darf höchstens ein AT END OF DATA-Statement verwendet werden.

Das AT END OF DATA-Statement wird nur dann ausgeführt, wenn die betreffende Schleife tatsächlich durchlaufen wird.

Referenzieren einer bestimmten Verarbeitungsschleife (r)

Normalerweise bezieht sich das Statement AT END OF DATA auf die jeweils äußerste aktive Verarbeitungsschleife. Wollen Sie, daß es sich auf eine andere aktive Schleife bezieht, so verwenden Sie hierzu die Notation “(r)”, wobei r das Statement-Label oder die Sourcecode-Zeilenummer des Statements ist, welches die gewünschte Schleife initiiert.

Feldwerte der Datenbankfelder

Zu dem Zeitpunkt, zu dem das AT END OF DATA-Statement ausgeführt wird, enthalten alle Datenbankfelder die Werte des zuletzt verarbeiteten Datensatzes.

Systemfunktionen

Natural-Systemfunktionen können in Verbindung mit einem AT END OF DATA-Statement verwendet werden, wie im Kapitel **Systemfunktionen** des *Natural-Referenzhandbuchs* beschrieben.

Beispiel

```

/* EXAMPLE 'AEDEXIS': AT END OF DATA (STRUCTURED MODE)
/*****
DEFINE DATA LOCAL
  1 EMPLOY-VIEW VIEW OF EMPLOYEES
    2 PERSONNEL-ID
    2 NAME
    2 FIRST-NAME
    2 SALARY (1)
    2 CURR-CODE (1)
END-DEFINE
/*****
LIMIT 5
EMP. FIND EMPLOY-VIEW WITH CITY = 'STUTTGART'
  IF NO RECORDS FOUND
    ENTER
  END-NOREC
  DISPLAY PERSONNEL-ID NAME FIRST-NAME
    SALARY (1) CURR-CODE (1)
/*
  ****
  AT END OF DATA
    IF *COUNTER (EMP.) = 0
      WRITE 'NO RECORDS FOUND'
      ESCAPE BOTTOM
    END-IF
    WRITE NOTITLE / 'SALARY STATISTICS;'
      / 7X 'MAXIMUM:' MAX(SALARY(1)) CURR-CODE (1)
      / 7X 'MINIMUM:' MIN(SALARY(1)) CURR-CODE (1)
      / 7X 'AVERAGE:' AVER(SALARY(1)) CURR-CODE (1)
  END-ENDDATA
/*
  ****
  END-FIND
/*****
END

```

PERSONNEL ID	NAME	FIRST-NAME	ANNUAL SALARY	CURRENCY CODE
11100328	BERGHAUS	ROSE	70800	DM
11100329	BARTHEL	PETER	42000	DM
11300313	AECKERLE	SUSANNE	55200	DM
11300316	KANTE	GABRIELE	61200	DM
11500304	KLUGE	ELKE	49200	DM
SALARY STATISTICS:				
	MAXIMUM:		70800	DM
	MINIMUM:		42000	DM
	AVERAGE:		55680	DM

Äquivalentes Reporting-Mode-Beispiel: siehe Programm AEDEX1R in Library SYSEXR.M.

AT END OF PAGE

Structured-Mode-Syntax

```
[AT] END [OF] PAGE [(rep)]
    statement...
END-ENDPAGE
```

Reporting-Mode-Syntax

```
[AT] END [OF] PAGE [(rep)]
{
    statement
    DO statement... DOEND
}
```

Verwandte Statements

AT TOP OF PAGE, DISPLAY, WRITE, INPUT, NEWPAGE.

Funktion

Mit dem AT END OF PAGE-Statement können Sie eine Verarbeitung angeben, die ausgeführt werden soll, wenn das Ende einer logischen Seite erreicht ist (“End-of-Page“-Bedingung; siehe Session-Parameter PS im *Natural-Referenzhandbuch*).

Eine “End-of-Page“-Bedingung kann auch aufgrund eines SKIP- oder NEWPAGE-Statements auftreten, nicht aber aufgrund eines EJECT- oder INPUT-Statements.

Ein AT END OF PAGE-Statement-Block wird nur ausgeführt, wenn das Objekt, das den Statement-Block enthält, zu dem Zeitpunkt, zu dem die “End-of-Page“-Bedingung auftritt, aktiv ist.

Ein AT END OF PAGE-Statement darf nicht in einer internen Subroutine stehen.

Dieses Statement ist non-prozedural (das heißt, seine Ausführung hängt von einem Ereignis ab, nicht davon, wo im Programm es steht).

Report-Spezifikation (*rep*)

Mit der Notation “(*rep*)” kann ein bestimmter anderer Report angegeben werden, auf den sich das Statement beziehen soll. Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden.

Falls nichts anderes angegeben wird, bezieht sich das AT END OF PAGE-Statement auf den ersten Report (Report 0).

Länge der logischen Seite

Da erst überprüft wird, ob eine “End-of-Page”-Bedingung besteht, nachdem ein DISPLAY- oder WRITE-Statement vollständig ausgeführt ist, kann es vorkommen, daß die von einem DISPLAY- oder WRITE-Statement erzeugte Ausgabe das Ende einer physischen Seite bereits überschritten hat, bevor eine “End-of-Page”-Bedingung entdeckt wird.

Um dies zu vermeiden und um sicherzustellen, daß über ein AT END OF PAGE-Statement ausgegebene Informationen wirklich am Ende einer physischen Ausgabeseite erscheint, muß die logische Seitenlänge (Session-Parameter PS) entsprechend kleiner als die Länge einer physischen Ausgabeseite gesetzt werden.

Letzte Seite

In einem Hauptprogramm ist eine “End-of-Page”-Bedingung auch dann gegeben, wenn die Ausführung des Programms durch ein ESCAPE-, STOP- oder END-Statement beendet wird.

In einer Subroutine gilt dies nicht; das heißt, ESCAPE, RETURN oder END lösen in einer Subroutine keine “End-of-Page”-Bedingung aus.

Systemfunktionen

Natural-Systemfunktionen können in Verbindung mit einem AT END OF DATA-Statement verwendet werden, wie im Kapitel **Systemfunktionen** des *Natural-Referenzhandbuchs* beschrieben.

Wenn eine Systemfunktion in einem AT END OF PAGE-Statement-Block verwendet wird, muß das betreffende DISPLAY-Statement eine GIVE SYSTEM FUNCTIONS-Klausel enthalten.

INPUT-Statement im AT END OF PAGE

Wenn Sie im AT END OF PAGE-Block ein INPUT-Statement verwenden, wird keine Seitenvorschub-Operation ausgeführt. Sie müssen in diesem Fall den Wert des Session-Parameters PS soweit reduzieren, daß die vom INPUT-Statement erzeugten Zeilen noch auf derselben physischen Seite Platz haben. Siehe auch INPUT-Statement (Split Screen Mode). Vgl. Beispiel 2.

Beispiel 1

```

/* EXAMPLE 'AEPEX1S': AT END OF PAGE (STRUCTURED MODE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 JOB-TITLE
  2 SALARY (1)
  2 CURR-CODE (1)
END-DEFINE
/*****
FORMAT PS=10
LIMIT 10
READ EMPLOY-VIEW BY PERSONNEL-ID FROM '20017000'
  DISPLAY NOTITLE GIVE SYSTEM FUNCTIONS
    NAME JOB-TITLE 'SALARY' SALARY(1) CURR-CODE (1)
/* ****
  AT END OF PAGE
    WRITE / 28T 'AVERAGE SALARY: ...' AVER(SALARY(1)) CURR-CODE (1)
  END-ENDPAGE
/* ****
END-READ
/*****
END

```

NAME	CURRENT POSITION	SALARY	CURRENCY CODE
CREMER	ANALYST	34000	USD
MARKUSH	TRAINEE	22000	USD
GEE	MANAGER	39500	USD
KUNEY	DBA	40200	USD
NEEDHAM	PROGRAMMER	32500	USD
JACKSON	PROGRAMMER	33000	USD
	AVERAGE SALARY: ...	33533	USD

Äquivalentes Reporting-Mode-Beispiel: siehe Programm AEPEX1R in Library SYSEXRM.

Beispiel 2

```

/* EXAMPLE 'AEPEX2': AT END OF PAGE WITH INPUT STATEMENT
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEEES
  2 NAME
  2 FIRST-NAME
  2 POST-CODE
  2 CITY
1 #START-NAME(A20)
END-DEFINE
/*****
FORMAT PS=21
REPEAT
  READ (15) EMPLOY-VIEW BY NAME = #START-NAME
  DISPLAY NOTITLE NAME FIRST-NAME POST-CODE CITY
END-READ
NEWPAGE
/*****
AT END OF PAGE
  MOVE NAME TO #START-NAME
  INPUT / '-' (79) / 10T 'Reposition to name ==>'
  #START-NAME (AD=MI) (''. ' to exit)'
  IF #START-NAME = '.'
    STOP
  END-IF
END-ENDPAGE
/*****
END-REPEAT
END

```

NAME	FIRST-NAME	POSTAL ADDRESS	CITY
ALEXANDER	STEPHEN	19711	NEWARK
ALEXANDER	GIL	21209	BALTIMORE
ALEXANDER	CHARLY	95616	DAVIS
ALEXANDER	HOLLY	53706	MADISON
ALLEGRE	KARL-OTTO	8100	CHARLEVILLE MEZIERES
ALLSOP	ALAN	DE3 3NL	DERBY
ALVAREZ	RAQUEL	28015	MADRID
AMOROS	FUENSANTA	28014	MADRID
ANDERSEN	ANITA	1850 V	KÖBENHAVN
ANDERSEN	KARIN	2720	VANLÖSE
ANDERSEN	LISSI	2650	HVIDOVRE
ANDERSON	JENNY	84112	SALT LAKE CITY
ANTLIFF	JANET	DE3 3EE	DERBY
ARCHER	ROBIN	DE4 8GR	DERBY
ARCONADA	ARANZAZU	28014	MADRID
Reposition to name ==> ARCONADA			('.' to exit)

AT START OF DATA

Structured-Mode-Syntax

```
[AT] START [OF] DATA [(r)]  
    statement...  
END-START
```

Reporting-Mode-Syntax

```
[AT] START [OF] DATA [(r)]  
    { statement }  
    DO statement... DOEND
```

Verwandte Statements

AT END OF DATA, FIND, READ, HISTOGRAM, SORT, READ WORK FILE.

Funktion

Mit dem Statement AT START OF DATA können Sie eine Verarbeitung angeben, die ausgeführt werden soll, unmittelbar nachdem der erste Datensatz einer mit READ, FIND, HISTOGRAM, SORT oder READ WORK FILE initiierten Verarbeitungsschleife gelesen worden ist. Falls das schleifeninitiiierende Statement eine WHERE-Klausel enthält, wird die AT START OF DATA-Verarbeitung erst dann ausgeführt, wenn der erste Datensatz gelesen wird, der sowohl das primäre Suchkriterium als auch die WHERE-Bedingung erfüllt.

Dieses Statement ist non-prozedural (das heißt, seine Ausführung hängt von einem Ereignis ab, nicht davon, wo im Programm es steht).

Feldwerte der Datenbankfelder

Zu dem Zeitpunkt, zu dem das AT START OF DATA-Statement ausgeführt wird, enthalten alle Datenbankfelder die Werte des zuerst verarbeiteten Datensatzes (d.h. des ersten Datensatzes, der die AT START OF DATA-Bedingung erfüllt).

Positionierung

Das AT START OF DATA-Statement muß *innerhalb* der betreffenden Verarbeitungsschleife stehen. Pro Verarbeitungsschleife darf höchstens ein AT START OF DATA-Statement verwendet werden.

Referenzieren einer bestimmten Verarbeitungsschleife (*r*)

Normalerweise bezieht sich das Statement AT START OF DATA auf die jeweils äußerste aktive Verarbeitungsschleife. Wollen Sie, daß es sich auf eine andere aktive Schleife bezieht, so verwenden Sie hierzu die Notation “(*r*)”, wobei *r* das Statement-Label oder die Sourcecode-Zeilenummer des Statements ist, welches die gewünschte Schleife initiiert.

Beispiel

```

/* EXAMPLE 'ASDEXIS': AT START OF DATA (STRUCTURED MODE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
1 #CNTL (A1) INIT <' '>
1 #CITY (A20) INIT <' '>
END-DEFINE
/*****
REPEAT
  INPUT 'ENTER VALUE FOR CITY' #CITY
  IF #CITY = ' ' OR = 'END'
    STOP
  END-IF
  FIND EMPLOY-VIEW WITH CITY = #CITY
  IF NO RECORDS FOUND
    WRITE NOTITLE NOHDR 'NO RECORDS FOUND'
    ESCAPE BOTTOM
  END-NOREC
/*****
  AT START OF DATA
    INPUT (AD=O) 'RECORDS FOUND' *NUMBER //
      'ENTER ''D'' TO DISPLAY RECORDS' #CNTL (AD=A)
    IF #CNTL NE 'D'
      ESCAPE BOTTOM
    END-IF
  END-START
/*****
  DISPLAY NAME FIRST-NAME
END-FIND
END-REPEAT
END

```

ENTER VALUE FOR CITY **PARIS**

RECORDS FOUND 24

ENTER 'D' TO DISPLAY RECORDS **D**

NAME	FIRST-NAME
MAIZIERE	ELISABETH
MARX	JEAN-MARIE
REIGNARD	JACQUELINE
RENAUD	MICHEL
REMOUE	GERMAINE
LAVENDA	SALOMON
BROUSSE	GUY
GIORDA	LOUIS
SIECA	FRANCOIS
CENSIER	BERNARD
DUC	JEAN-PAUL
CAHN	RAYMOND
MAZUY	ROBERT
VALLY	ALAIN
BRETON	JEAN-MARIE
GIGLEUX	JACQUES
XOLIN	CHRISTIAN
LEGRIS	ROGER
RIVIERE	JEAN-LUC
REICH	MARC
VVVV	

Äquivalentes Reporting-Mode-Beispiel: siehe Programm ASDEX1R in Library SYSEXRM.

AT TOP OF PAGE

Structured-Mode-Syntax

```
[AT] TOP [OF] PAGE [(rep)]  
    statement...  
END-TOPPAGE
```

Reporting-Mode-Syntax

```
[AT] TOP [OF] PAGE [(rep)]  
    { statement  
      DO statement... DOEND }
```

Verwandte Statements

AT END OF PAGE, NEWPAGE.

Funktion

Mit dem AT TOP OF PAGE-Statement können Sie eine Verarbeitung angeben, die ausgeführt werden soll, wenn eine neue Seite beginnt.

Eine neue Seite beginnt, wenn entweder die ausgegebene Zeilenzahl die mit dem Session-Parameter PS gesetzte Seitenlänge überschreitet oder ein NEWPAGE-Statement ausgeführt wird. Ein EJECT-Statement führt ebenfalls zu einem Seitenvorschub, löst aber keine AT TOP OF PAGE-Verarbeitung aus.

Ein AT TOP OF PAGE-Statement-Block wird nur ausgeführt, wenn das Objekt, das den Statement-Block enthält, zu dem Zeitpunkt, zu dem die “Top-of-Page”-Bedingung auftritt, aktiv ist.

Erzeugt ein AT TOP OF PAGE-Statement eine Ausgabe, so wird diese unter der Seitentitelzeile ausgegeben, wobei zwischen beiden automatisch eine Leerzeile ausgegeben wird.

Dieses Statement ist non-prozedural (das heißt, seine Ausführung hängt von einem Ereignis ab, nicht davon, wo im Programm es steht).

Einschränkung

Ein AT TOP OF PAGE-Statement darf nicht in einer internen Subroutine stehen.

Report-Spezifikation (*rep*)

Mit der Notation “(*rep*)” kann ein bestimmter anderer Report angegeben werden, auf den sich das Statement beziehen soll. Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden.

Falls nichts anderes angegeben wird, bezieht sich das AT TOP OF PAGE-Statement auf den ersten Report (Report 0).

Beispiel

```

/* EXAMPLE 'ATPEX1S': AT TOP OF PAGE (STRUCTURED MODE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
  2 DEPT
END-DEFINE
/*****
FORMAT PS=15
LIMIT 15
READ EMPLOY-VIEW BY NAME STARTING FROM 'L'
  DISPLAY 2X NAME 4X FIRST-NAME CITY DEPT
  WRITE TITLE UNDERLINED 'EMPLOYEE REPORT'
  WRITE TRAILER '-' (78)
/*****
  AT TOP OF PAGE
  WRITE 'BEGINNING NAME:' NAME
  END-TOPPAGE
/*****
  AT END OF PAGE
  SKIP 1
  WRITE 'ENDING NAME: ' NAME
  END-ENDPAGE
END-READ
END

```

EMPLOYEE REPORT			
BEGINNING NAME: LAFON NAME	FIRST-NAME	CITY	DEPARTMENT CODE
LAFON	CHRISTIANE	PARIS	VENT18
LANDMANN	HARRY	ESCHBORN	MARK29
LANE	JACQUELINE	DERBY	MGMT02
LANKATILLEKE	LALITH	FRANKFURT	PROD22
LANNON	BOB	LINCOLN	SALE20
LANNON	LESLIE	SEATTLE	SALE30
LARSEN	CARL	FARUM	SYSA01
LARSEN	MOGENS	VEMMELEV	SYSA02

ENDING NAME: LARSEN

Äquivalentes Reporting-Mode-Beispiel: siehe Programm ATPEX1R in Library SYSEXRM.

BACKOUT TRANSACTION

BACKOUT [TRANSACTION]

Verwandtes Statement

END TRANSACTION.

Funktion

Dieses Statement bewirkt, daß alle Datenbankänderungen, die während der laufenden, noch nicht abgeschlossenen logischen Transaktion ausgeführt wurden, rückgängig gemacht werden; außerdem bewirkt es, daß alle während der Transaktion gehaltenen Datensätze wieder freigegeben werden.

Das BACKOUT TRANSACTION-Statement wird nur ausgeführt, wenn eine Datenbanktransaktion unter Kontrolle von Natural stattgefunden hat. Für welche Datenbanken das Statement ausgeführt wird, hängt davon ab, wie der Profilparameter ET (vgl. *Natural-Referenzhandbuch* und *Natural Installation and Operations Documentation*) gesetzt ist:

- Ist ET=OFF gesetzt, wird das Statement nur für die von der Transaktion betroffene Datenbank ausgeführt.
- Ist ET=ON gesetzt, wird das Statement für alle Datenbanken ausgeführt, die seit der letzten Ausführung eines BACKOUT TRANSACTION- oder END TRANSACTION-Statements referenziert wurden.

Hinweis für Entire System Server:

Mit Entire System Server kann dieses Statement nicht verwendet werden.

Hinweise für DL/I-Datenbanken

Da die PSB-Initialisierung durch eine "Syncpoint"-Anfrage beendet wird, speichert Natural die PSB-Position, bevor das BACKOUT TRANSACTION-Statement ausgeführt wird. Bevor das nächste Kommando ausgeführt wird, re-initialisiert Natural den PSB und versucht, die PCB-Position so zu setzen, wie sie vor dem BACKOUT TRANSACTION-Statement war. Die PCB-Position kann vorverlegt werden, falls in der Zeit zwischen BACKOUT TRANSACTION und dem nachfolgenden Kommando ein adressiertes Segment gelöscht wurde.

Hinweise für SQL-Datenbanken

Da die meisten SQL-Datenbanken bei Beendigung einer logischen Arbeitseinheit alle Cursor schließen, darf ein BACKOUT TRANSACTION-Statement nicht innerhalb einer datenbankverändernden Verarbeitungsschleife stehen, sondern muß nach einer solchen plaziert werden.

BACKOUT TRANSACTION über Abbruchkommando

Unterbricht der Benutzer mit einem Natural-Terminalkommando (Kommando “%%” oder CLEAR-Taste) eine gerade aktive Natural-Operation, dann führt Natural ein BACKOUT TRANSACTION-Statement aus (weitere Informationen siehe Terminalkommando “%%” im *Natural-Referenzhandbuch*).

Weitere Informationen

Weitere Informationen zur Natural-Transaktionslogik und zum Beenden/Abbrechen einer logischen Transaktion finden Sie im Kapitel **Datenbankzugriffe** des *Natural Leitfadens zur Programmierung*.

Beispiel

```

/* EXAMPLE 'BOTEX1S': BACKOUT TRANSACTION (STRUCTURED MODE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 DEPT
  2 LEAVE-DUE
  2 LEAVE-TAKEN
1 #DEPT (A6)
1 #RESP (A3)
END-DEFINE
/*****
LIMIT 3
INPUT 'DEPARTMENT TO BE UPDATED:' #DEPT
  IF #DEPT = ' '
    STOP
  END-IF
/*****
FIND EMPLOY-VIEW WITH DEPT = #DEPT
  IF NO RECORDS FOUND
    REINPUT 'NO RECORDS FOUND'
  END-NOREC
  INPUT 'NAME:          ' NAME (AD=O) /
    'LEAVE DUE:        ' LEAVE-DUE (AD=M) /
    'LEAVE TAKEN:' LEAVE-TAKEN (AD=M)

  UPDATE
END-FIND
/*****
INPUT 'UPDATE TO BE PERFORMED YES/NO:' #RESP
  DECIDE ON FIRST #RESP
    VALUE 'YES'
      END TRANSACTION
    VALUE 'NO'
      BACKOUT TRANSACTION
    NONE
      REINPUT 'PLEASE ENTER YES OR NO'
  END-DECIDE
/*****
END

```

DEPARTMENT TO BE UPDATED: **MGMT30**

NAME: POREE
LEAVE DUE: 45
LEAVE TAKEN: 31

UPDATE TO BE PERFORMED YES/NO: **NO**

Äquivalentes Reporting-Mode-Beispiel: siehe Programm BOTEX1R in Library SYSEXR.M.

BEFORE BREAK PROCESSING

Structured-Mode-Syntax

```
BEFORE [BREAK] [PROCESSING]
    statement...
END-BEFORE
```

Reporting-Mode-Syntax

```
BEFORE [BREAK] [PROCESSING]
    { statement }
    DO statement... DOEND
```

Verwandtes Statement

AT BREAK.

Funktion

Das Statement `BEFORE BREAK PROCESSING` wird im Zusammenhang mit einem automatischen Gruppenwechsel verwendet, und zwar um Verarbeitungen anzugeben, die ausgeführt werden sollen:

- bevor der Wert des Gruppenwechsel-Kontrollfeldes geprüft wird;
- bevor ein `AT BREAK`-Statement-Block ausgeführt wird
- bevor Natural-Systemfunktionen ausgewertet werden.

Meistens wird `BEFORE BREAK PROCESSING` eingesetzt, um Benutzervariablen zu initialisieren oder zu berechnen, die bei einer anschließenden Gruppenwechsel-Verarbeitung (siehe `AT BREAK`-Statement) benutzt werden sollen.

Wird keine an einen Gruppenwechsel geknüpfte Verarbeitung ausgeführt (d.h. wenn die betreffende Verarbeitungsschleife kein `AT BREAK`-Statement enthält), so wird der `BEFORE BREAK PROCESSING`-Statement-Block *nicht* ausgeführt.

Dieses Statement ist non-prozedural (das heißt, seine Ausführung hängt von einem Ereignis ab, nicht davon, wo im Programm es steht).

Einschränkungen

Das Statement `BEFORE BREAK PROCESSING` kann nur in Verbindung mit einer Verarbeitungsschleife verwendet werden, die mit `FIND`, `READ`, `HISTOGRAM`, `SORT` oder `READ WORK FILE` initiiert wird. In einer Verarbeitungsschleife darf höchstens ein `BEFORE BREAK PROCESSING`-Statement stehen. Das Statement darf an beliebiger Stelle innerhalb einer Schleife stehen und bezieht sich immer auf die Schleife, in der es steht.

Ein `BEFORE BREAK PROCESSING`-Statement darf nicht in Verbindung mit einem `PERFORM BREAK PROCESSING`-Statement verwendet werden.

Beispiel

```

/* EXAMPLE 'BBPEX1': BEFORE BREAK PROCESSING
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 SALARY (1)
  2 BONUS (1,1)
1 #INCOME (P11)
END-DEFINE
/*****
LIMIT 7
READ EMPLOY-VIEW BY CITY = 'L'
/*****
  BEFORE BREAK PROCESSING
    COMPUTE #INCOME = SALARY (1) + BONUS (1,1)
  END-BEFORE
/*****
  AT BREAK OF CITY
    WRITE NOTITLE 'AVERAGE INCOME FOR' OLD (CITY) 20X AVER(#INCOME) /
  END-BREAK
/*****
  DISPLAY CITY 'NAME' NAME 'SALARY' SALARY (1) 'BONUS' BONUS (1,1)
END-READ
END

```

CITY	NAME	SALARY	BONUS
LA BASSEE	HULOT	165000	70000
AVERAGE INCOME FOR LA BASSEE			235000
LA CHAPELLE ST LUC	GUILLARD	124100	23000
LA CHAPELLE ST LUC	BERGE	198500	50000
LA CHAPELLE ST LUC	POLETTE	124090	23000
LA CHAPELLE ST LUC	DELAUNEY	115000	23000
LA CHAPELLE ST LUC	SCHECK	125600	23000
LA CHAPELLE ST LUC	KREEBS	184550	50000
AVERAGE INCOME FOR LA CHAPELLE ST LUC			177306

CALL

CALL [INTERFACE4] *operand1* [USING] [*operand2*]... 128

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	ja	nein
Operand2	C S A G	A N P I F B D T L C G	ja	ja

- CALL auf Großrechnern
- Teil I: CALL unter OpenVMS, UNIX und Windows
- INTERFACE4
- Teil II: CALL unter OpenVMS, UNIX und Windows.

CALL auf Großrechnern

Funktion

Mit dem CALL-Statement können Sie von einem Natural-Programm aus ein anderes, in einer anderen Standard-Programmiersprache geschriebenes Programm aufrufen, wobei im Anschluß daran die Verarbeitung des Natural-Programms mit dem nächsten Statement nach dem CALL-Statement fortgesetzt wird.

Das aufgerufene Programm kann in einer beliebigen anderen Programmiersprache, die eine Standard-CALL-Schnittstelle unterstützt, geschrieben sein. Mehrere CALL-Statements können verwendet werden, um ein Programm mehrmals oder mehrere Programme aufzurufen.

Ein CALL-Statement kann auch in einem Programm, das unter Kontrolle eines TP-Monitors ausgeführt werden soll, angegeben werden, vorausgesetzt der TP-Monitor unterstützt eine CALL-Schnittstelle.

Programmname (*operand1*)

Der Name des aufgerufenen Programms (*operand1*) kann entweder als Konstante angegeben werden oder — falls je nach Programmlogik verschiedene Programme aufgerufen werden sollen — als alphanumerische Variable mit Länge 1 bis 8. Ein Programmname muß linksbündig in der Variablen stehen.

Parameter (*operand2*)

Das CALL-Statement kann bis zu 128 Parameter (*operand2*) enthalten, es sei denn, die INTERFACE4-Option wird verwendet. In diesem Fall gibt es keine Beschränkung, wie viele Parameter benutzt werden dürfen. Hierbei gelten die Standard-Linkage-Registerkonventionen. Für jedes angegebene Parameterfeld wird in der Parameterliste eine Adresse übergeben.

Wird ein Gruppenname verwendet, so wird die Gruppe in einzelne Felder umgesetzt, d.h. der Benutzer muß das erste Feld der Gruppe angeben, falls er die Anfangsadresse einer Gruppe spezifizieren will.

Anmerkung:

Die interne Darstellung der positiven Vorzeichen gepackter Zahlen ändert sich auf den vom PSIGNF-Parameter des NTCMPO-Makros angegebenen Wert, bevor die Kontrolle an das externe Programm übergeben wird.

Return Code

Um den Condition Code eines aufgerufenen Programms (Inhalt von Register 15 beim Rücksprung zum Natural-Programm) zu erhalten, können Sie die Natural-Systemfunktion RET verwenden.

Beispiel:

```
...  
RESET #RETURN(B4)  
CALL 'PROG1'  
IF RET ('PROG1') > #RETURN  
    WRITE 'ERROR OCCURRED IN PROGRAM1'  
END-IF  
...
```

Registerbelegung

Register	Inhalt
R1	Adresse der Parameteradressenliste.
R2	Adresse der Feld-(Parameter-)Beschreibungsliste. Die Feldbeschreibungsliste enthält Informationen über die ersten 128 in der Parameterliste übergebenen Felder. Jede Beschreibung ist ein 4 Byte langer Eintrag, der sich wie folgt zusammensetzt: – 1. Byte: Variablentyp (A, B, ...) Falls das Feld vom Typ "N" oder "P" ist: – 2. Byte: Anzahl der Stellen insgesamt; – 3. Byte: Anzahl der Stellen vor dem Komma (Dezimalpunkt); – 4. Byte: Anzahl der Stellen nach dem Komma (Dezimalpunkt). Bei allen anderen Feldtypen: – 2. Byte: ungenutzt; – 3.–4. Byte: Länge des Feldes.
R3	Adresse der Feldlängen-Liste. Diese Liste enthält die Längen der in der Parameterliste übergebenen Felder. Bei einem Array ergibt sich die Länge aus der Summe der Längen der einzelnen Ausprägungen.
R13	Adresse des 18-Wort-Speicherbereiches.
R14	Rücksprungadresse.
R15	Eingangsadresse/Returncode.

Begrenzungszuordnung

Der Natural-Datenbereich, in dem alle Benutzervariablen gespeichert sind, beginnt immer mit einer Doppelwort-Begrenzung.

Wird DEFINE DATA verwendet, sind alle Datenblöcke (z.B. LOCAL- oder GLOBAL-Blöcke) doppelwort-begrenzt und alle Strukturen (Level 1) vollwort-begrenzt.

Die Begrenzungszuordnung innerhalb des Datenbereichs ist Sache des Benutzers und richtet sich nach der Reihenfolge, in der die Variablen für Natural definiert sind.

Adabas-Aufrufe

Ein aufgerufenes Programm darf einen Adabas-Aufruf beinhalten. Das aufgerufene Programm darf kein Adabas-Open- oder -Close-Kommando absetzen. Adabas öffnet alle angesprochenen Dateien. Soll Adabas-EXU-Modus (EXclusive Update) verwendet werden, muß zum Öffnen aller angesprochenen Dateien der Natural-Profilparameter OPRB benutzt werden; bevor Sie den EXU-Update-Modus einsetzen, sollten Sie in jedem Fall Ihren Natural-Administrator konsultieren.

Direktes/Dynamisches Laden

Das aufgerufene Programm kann entweder direkt an den Natural-Nukleus gelinkt werden (indem es mit dem CSTATIC-Parameter im Natural-Parametermodul angegeben wird; vgl. *Natural-Referenzhandbuch*), oder es kann dynamisch geladen werden, wenn es zum erstenmal aufgerufen wird.

Soll es dynamisch geladen werden, so muß die Lademodul-Library, die das aufgerufene Programm enthält, mit der Natural-Lade-Library verkettet werden, und zwar entweder in der Natural-Ausführungs-JCL oder in der entsprechenden Programm-Library des TP-Monitors. Weitere Einzelheiten erfragen Sie bitte bei Ihrem Natural-Administrator.

Beispiel

Das Beispiel auf der nächsten Seite zeigt ein Natural-Programm, welches das COBOL-Programm "TABSUB" aufruft, und zwar zu dem Zweck, Ländercodes (COUNTRY-CODE) in die entsprechenden Ländernamen (COUNTRY-NAME) umzusetzen. Das Natural-Programm übergibt zwei Parameter an das COBOL-Programm: der erste Parameter ist der Ländercode, so wie er von der Datenbank gelesen wird; der zweite Parameter dient dazu, den Ländernamen zurückzugeben.

Aufrufendes Natural-Programm:

```
* EXAMPLE 'CALEX1': CALL PROGRAM 'TABSUB'
* *****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 BIRTH
  2 COUNTRY
1 #COUNTRY      (A3)
1 #COUNTRY-NAME (A15)
END-DEFINE
*
MOVE EDITED '19550701' TO #FIND-FROM (EM=YYYYMMDD)
MOVE EDITED #19550731' TO #FIND-TO   (EM=YYYYMMDD)
*
FIND EMPLOY-VIEW WITH BIRTH = #FIND-FROM THRU #FIND-TO
  MOVE COUNTRY TO #COUNTRY
  CALL 'TABSUB' #COUNTRY #COUNTRY-NAME
  DISPLAY NAME BIRTH (EM=YYYY-MM-DD) #COUNTRY-NAME
END-FIND
END
```

Aufgerufenes COBOL-Programm "TABSUB":

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TABSUB.
REMARKS. THIS PROGRAM PROVIDES THE COUNTRY NAME
        FOR A GIVEN COUNTRY CODE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
01 COUNTRY-CODE PIC X(3).
01 COUNTRY-NAME PIC X(15).
PROCEDURE DIVISION USING COUNTRY-CODE COUNTRY-NAME.
P-CONVERT.
    MOVE SPACES TO COUNTRY-NAME.
    IF COUNTRY-CODE = 'BLG' MOVE 'BELGIUM' TO COUNTRY-NAME.
    IF COUNTRY-CODE = 'DEN' MOVE 'DENMARK' TO COUNTRY-NAME.
    IF COUNTRY-CODE = 'FRA' MOVE 'FRANCE' TO COUNTRY-NAME.
    IF COUNTRY-CODE = 'GER' MOVE 'GERMANY' TO COUNTRY-NAME.
    IF COUNTRY-CODE = 'HOL' MOVE 'HOLLAND' TO COUNTRY-NAME.
    IF COUNTRY-CODE = 'ITA' MOVE 'ITALY' TO COUNTRY-NAME.
    IF COUNTRY-CODE = 'SPA' MOVE 'SPAIN' TO COUNTRY-NAME.
    IF COUNTRY-CODE = 'UK'  MOVE 'UNITED KINGDOM' TO COUNTRY-NAME.
P-RETURN.
GOBACK.
```

Verbindungskonventionen

Im Batch-Betrieb wird “Standard Linkage Register Notation” verwendet. Für jeden TP-Monitor gelten hierbei andere Konventionen. Diese Konventionen müssen unbedingt befolgt werden, da es sonst zu unvorhersehbaren Ergebnissen kommen kann. Die für die einzelnen unterstützten TP-Monitore gültigen Konventionen sind im folgenden beschrieben.

CALL unter Com-plete

Das aufgerufene Programm muß sich in der Com-plete-Online-Lade-Library befinden, damit Com-plete es dynamisch laden kann. Zum Katalogisieren des Programms kann die Com-plete-Utility ULIB verwendet werden.

CALL unter CICS

Das aufgerufene Programm muß sich entweder in der DFHRPL-Library befinden oder in der Lademodul-Library, die mit der CICS-Library verkettet ist. Damit CICS das Programm finden und laden kann, muß das Programm außerdem in der aktiven PPT eingetragen sein.

Die Linkage-Konvention übergibt die Parameteradressenliste sowie die Feldbeschreibungsliste in den ersten Vollwörtern der TWA und der COMMAREA. Der Parameter FLDLEN im Parametermodul NCIPARM bestimmt, ob die Feldlängenliste auch übergeben wird (standardmäßig wird sie nicht übergeben).

An der Länge der COMMAREA (8 oder 12) läßt sich erkennen, wieviele Listenadressen (2 oder 3) übergeben werden. Die letzte Listenadresse wird dadurch angezeigt, daß das linke Bit gesetzt ist.

Es ist Sache des Benutzers, die Adressierbarkeit der TWA bzw. der COMMAREA zu gewährleisten, falls das aufgerufene Programm für Natural nicht als statisches oder direkt gelinktes Programm definiert ist; in diesem Fall werden die Adresse der Parameterliste über Register 1, die Adresse der Feldbeschreibungsliste über Register 2 und die Adresse der Feldlängenliste in Register 3 übergeben.

Wenn Sie statt der Adresse der Parameteradressenliste die Parameterwerte selbst in der COMMAREA übergeben möchten, führen Sie vor dem Aufruf das Terminalkommando %P=C aus (siehe *Natural Referenzhandbuch*).

Wenn ein Natural-Programm ein Nicht-Natural-Programm aufruft und das aufgerufene Programm im Dialog einen Terminal-I/O absetzt, ist der Natural-Thread normalerweise solange blockiert, bis der Benutzer eine Eingabe gemacht hat. Um zu vermeiden, daß der Thread blockiert ist, können Sie das Terminalkommando %P=V verwenden (siehe *Natural Referenzhandbuch*).

Wenn ein Natural-Programm unter CICS ein Nicht-Natural-Programm aufruft, geschieht der Aufruf normalerweise über eine "EXEC CICS LINK"-Anforderung. Wenn für den Aufruf stattdessen Standard Linkage verwendet werden soll, geben Sie das Terminalkommando %P=S ein (siehe *Natural Referenzhandbuch*). Voraussetzung ist, daß das aufgerufene Programm den Standard-Linkage-Konventionen mit Standard-Registerverwendung entspricht.

In 31-Bit-Modus-Umgebungen gilt folgendes: wenn ein mit AMODE=24 gelinktes Programm aufgerufen wird und die Threads liegen über 16 MB, wird automatisch ein Wertaufruf (call by value) durchgeführt, d.h. die angegebenen Parameter, die an das aufgerufene Programm übergeben werden sollen, werden in den Bereich unterhalb 16 MB kopiert.

Returncodes unter CICS

CICS selbst unterstützt zwar keine Condition Codes für einen Aufruf mit CICS-Konventionen (EXEC CICS LINK); aber das Natural/CICS-Interface unterstützt Returncodes für das CALL-Statement: wenn die Kontrolle vom aufgerufenen Programm zurückgegeben wird, überprüft Natural, ob sich das erste Vollwort der COMMAREA geändert hat.

Ist dies der Fall, wird dessen neuer Inhalt als Returncode genommen. Hat es sich nicht geändert, wird das erste Vollwort der TWA geprüft und dessen neuer Inhalt als Returncode genommen. Hat sich keins der beiden Vollwörter geändert, ist der Returncode "0".

Anmerkung:

Wenn die Parameterwerte in der COMMAREA übergeben werden (%P=C), ist der Returncode immer "0".

Beispiel unter CICS:

```

IDENTIFICATION DIVISION.
PROGRAM-ID. TABSUB.
REMARKS. THIS PROGRAM PERFORMS A TABLE LOOK-UP AND
        RETURNS A TEXT MESSAGE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 MSG-TABLE.
   03 FILLER          PIC X(15) VALUE 'MESSAGE1'   '.
   03 FILLER          PIC X(15) VALUE 'MESSAGE2'   '.
   03 FILLER          PIC X(15) VALUE 'MESSAGE3'   '.
   03 FILLER          PIC X(15) VALUE 'MESSAGE4'   '.
   03 FILLER          PIC X(15) VALUE 'MESSAGE5'   '.
   03 FILLER          PIC X(15) VALUE 'MESSAGE6'   '.
   03 FILLER          PIC X(15) VALUE 'MESSAGE7'   '.
01 TAB REDEFINES MSG-TABLE.
   03 MESSAGE OCCURS 7 TIMES PIC X(15).
LINKAGE SECTION.
01 TWA-DATA.
   03 PARM-POINTER   USAGE IS POINTER.
01 PARM-LIST.
   03 DATA-LOC-IN   USAGE IS POINTER.
   03 DATA-LOC-OUT  USAGE IS POINTER.
01 INPUT-DATA.
   03 INPUT-NUMBER   PIC 99.
01 OUTPUT-DATA.
   03 OUTPUT-MESSAGE PIC X(15).
PROCEDURE DIVISION.
100-INIT.
   EXEC CICS ADDRESS TWA(ADDRESS OF TWA-DATA) END-EXEC.
   SET ADDRESS OF PARM-LIST   TO PARM-POINTER.
   SET ADDRESS OF INPUT-DATA  TO DATA-LOCIN.
   SET ADDRESS OF OUTPUT DATA TO DATA-LOC-OUT.
200-PROCESS.
   MOVE MESSAGE (INPUT-NUMBER) TO OUTPUT-MESSAGE.
300-RETURN.
   EXEC CICS RETURN END-EXEC.
400-DUMMY.
   GO-BACK.

```

Anmerkung:

Bei einigen Versionen des COBOL-Compilers, die keine SERVICE RELOAD-Statements verwenden, sollten diese Statements weggelassen werden.

Aufrufen eines PL/I-Programms

Ist das aufgerufene Programm in PL/I geschrieben, erfordert dies zusätzlich folgendes:

- Wenn das Programm link-editiert wird, muß ein ENTRY PLICALLA-Statement zur Verfügung gestellt werden. Dieses Statement bewirkt, daß das PL/I-Lademodul die Kontrolle als Unterprogramm (d.h. als aufgerufenes Programm) erhält.
Wenn das Programm rekursiv aufgerufen werden soll, können Sie auch das Programm NATPLICA aus der Natural-Source-Library verwenden. NATPLICA ist ein Beispiel dafür, wie ein PL/I-Programm von einem Natural-Programm aus rekursiv aufgerufen werden kann, ohne Speicherengpässe zu verursachen (Näheres hierzu entnehmen Sie bitte den Kommentaren in NATPLICA selbst). Eine Beschreibung des Statements ENTRY PLICALLA und weitere Informationen zum Aufruf eines PL/I-Programms finden Sie in der entsprechenden IBM-PL/I-Dokumentation.
- Da die Parameterliste eine Standardliste und keine von einem anderen PL/I-Programm übergebene Argumentliste ist, zeigen die übergebenen Adressen nicht auf einen LOCAL DESCRIPTOR. Dieses Problem löst man, indem man die Parameterfelder als arithmetische Variablen definiert. Dies bewirkt, daß PL/I die Parameterliste als Adressen von Daten und nicht als Adressen von LOCAL-DESCRIPTOR-Kontrollblöcken behandelt.

Es wird empfohlen, die Parameterfelder folgendem Beispiel entsprechend zu definieren:

```
PLIPROG: PROC(INPUT_PARM_1, INPUT_PARM_2) OPTIONS(MAIN);
  DECLARE (INPUT_PARM_1, INPUT_PARM_2) FIXED;
  PTR_PARM_1 = ADDR(INPUT_PARM_1);
  PTR_PARM_2 = ADDR(INPUT_PARM_2);
  DECLARE FIRST_PARM      PIC '99'   BASED (PTR_PARM_1);
  DECLARE SECOND_PARM     CHAR(12)   BASED (PTR_PARM_2);
```

Jeder Parameter der Input-Liste sollte als eindeutiges Element behandelt werden. Die Anzahl der Eingabeparameter sollte mit der Zahl der vom Natural-Programm übergebenen genau übereinstimmen. Die Eingabeparameter und ihre Attribute müssen den Natural-Definitionen entsprechen, andernfalls kann es zu unvorhersehbaren Ergebnissen kommen. Weitere Informationen zur Übergabe von Parametern an PL/I-Programme finden Sie in der entsprechenden IBM-PL/I-Dokumentation.

Beispiel für den Aufruf eines PL/I-Programms:

```

/* EXAMPLE 'CALEX2': CALL PROGRAM 'NATPLI'
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 AREA-CODE
  2 REDEFINE AREA-CODE
  3 #AC(N1)
1 #INPUT-NUMBER (N2)
1 #OUTPUT-COMMENT (A15)
END-DEFINE
/*****
READ EMPLOY-VIEW IN LOGICAL SEQUENCE BY NAME
      STARTING FROM 'WAGNER'
      MOVE ' ' TO #OUTPUT-COMMENT
      MOVE #AC TO #INPUT-NUMBER
CALL 'NATPLI' #INPUT-NUMBER #OUTPUT-COMMENT
END-READ
END

NATPLI: PROC(PARM_COUNT, PARM_COMMENT) OPTIONS(MAIN);
/* */
/* THIS PROGRAM ACCEPTS AN INPUT NUMBER */
/* AND TRANSLATES IT TO AN OUTPUT CHARACTER */
/* STRING FOR PLACEMENT ON THE FINAL */
/* NATURAL REPORT */
/* */
/* */
DECLARE PARM_COUNT, PARM_COMMENT FIXED;
DECLARE ADDR BUILTIN;
COUNT_PTR = ADDR(PARM_COUNT);
COMMENT_PTR = ADDR(PARM_COMMENT);
DECLARE INPUT_NUMBER PIC '99' BASED (COUNT_PTR);
DECLARE OUTPUT_COMMENT CHAR(15) BASED (COMMENT_PTR);
DECLARE COMMENT_TABLE(9) CHAR(15) STATIC INITIAL
('COMMENT1 ',
 'COMMENT2 ',
 'COMMENT3 ',
 'COMMENT4 ',
 'COMMENT5 ',
 'COMMENT6 ',
 'COMMENT7 ',
 'COMMENT8 ',
 'COMMENT9 ');
OUTPUT_COMMENT = COMMENT_TABLE(INPUT_NUMBER);
RETURN;
END NATPLI;

```

Beispiel für den Aufruf eines PL/I-Programms, das unter CICS abläuft:

```
/* EXAMPLE 'CALEX3': CALL PROGRAM 'CICSP'
/*****
DEFINE DATA LOCAL
1 #MESSAGE (A10) INIT <' '>
END-DEFINE
/*****
CALL 'CICSP' #MESSAGE
DISPLAY #MESSAGE
/*****
END

CICSP: PROCEDURE OPTIONS (MAIN REENTRANT);
      DCL 1          TWA_ADDRESS    BASED(TWA_POINTER);
          2          LIST_ADDRESS  POINTER;
      DCL 1 PTR_TO_LIST            BASED(LIST_ADDRESS);
          2 PARM_01                POINTER;
      DCL MESSAGE CHAR(10) BASED(PARM_01);
      EXEC CICS ADDRESS TWA(TWA_POINTER);
      MESSAGE='SUCCESS'; EXEC CICS RETURN; END CICSP;
```

Teil I: CALL unter OpenVMS, UNIX und Windows

Funktion

Mit dem CALL-Statement können Sie von einem Natural-Programm aus eine andere, in einer anderen Standard-Programmiersprache geschriebene Funktion aufrufen, wobei anschließend die Verarbeitung des Natural-Programms mit dem nächsten Statement nach dem CALL-Statement fortgesetzt wird.

Die aufgerufene Funktion kann in einer beliebigen anderen Programmiersprache, die eine Standard-CALL-Schnittstelle unterstützt, geschrieben sein. Es können mehrere CALL-Statements verwendet werden, um eine Funktion mehrmals oder mehrere Funktionen aufzurufen.

Name der aufgerufenen Funktion (*operand1*)

Der Name der aufgerufenen Funktion (*operand1*) kann entweder als Konstante angegeben werden oder — falls je nach Programmlogik verschiedene Funktionen aufgerufen werden sollen — als alphanumerische Variable mit Länge 1 bis 8. Ein Funktionsname muß linksbündig in der Variablen stehen.

Parameter (*operand2*)

Das CALL-Statement kann bis zu 128 Parameter (*operand2*) enthalten. Für jedes angegebene Parameterfeld wird in der Parameterliste eine Adresse an die aufgerufene Funktion übergeben.

Wird ein Gruppenname verwendet, so wird die Gruppe in einzelne Felder umgesetzt, d.h. der Benutzer muß das erste Feld der Gruppe angeben, falls er die Anfangsadresse einer Gruppe spezifizieren will.

Anmerkung:

Wenn eine anwendungsunabhängige Variable (AIV) oder Kontext-Variable als Parameter an einen User Exit übergeben wird, gelten die folgenden Beschränkungen: wenn der User Exit ein Natural-Subprogramm aufruft, das eine neue AIV oder Kontext-Variable erstellt, dann ist der Parameter nach Rückkehr vom Subprogramm ungültig. Dies gilt ungeachtet der Tatsache, ob die neue AIV/Kontext-Variable vom Subprogramm selbst oder von einem anderen, direkt oder indirekt vom Subprogramm aufgerufenen Objekt erstellt wird.

INTERFACE4

- INTERFACE4 — Externe 3GL-Programmierschnittstelle
- Operanden-Struktur für INTERFACE4
- INTERFACE4-Parameterzugriff
- Exportierte Funktionen.

Anmerkung:

Die Option INTERFACE4 ist auf Großrechnern nicht verfügbar.

Das Schlüsselwort INTERFACE4 gibt den Typ der Schnittstelle an, die zum Aufruf des externen Programms verwendet wird. Dieses Schlüsselwort ist optional. Wenn dieses Schlüsselwort angegeben wird, wird die als INTERFACE4 definierte Schnittstelle zum Aufruf des externen Programms verwendet. Die folgende Tabelle führt die Unterschiede zwischen dem mit INTERFACE4 benutzten CALL-Statement und dem ohne INTERFACE4 benutzten auf.

	CALL-Statement ohne Schlüsselwort INTERFACE4	CALL-Statement mit Schlüsselwort INTERFACE4
Anzahl der möglichen Parameter	128	unbegrenzt (32767)
Maximale Länge eines Parameters	64 K	1 GB
Array-Informationen einlesen	nein	ja
Unterstützung großer und dynamischer Operanden	nein	ja
Parameter-Zugriff über API	nein	ja

INTERFACE4 — Externe 3GL-Programmierschnittstelle

Die Schnittstelle des externen 3GL-Programms wird wie folgt definiert, wenn INTERFACE4 im Natural CALL-Statement angegeben wird:

NATFCT funktionname (numparm, parmhandle, traditional)

USR-WORD	*numparm;	16 Bit umfassender Kurzwert ohne Vorzeichen, der die Gesamtzahl der übertragenen Operanden (operand2) enthält
void	*parmhandle;	Adresse der Parameterübergabe-Struktur
void	*traditional;	Schnittstellen-Typ prüfen (wenn es keine NULL-Adresse ist, handelt es sich um die traditionelle CALL-Schnittstelle)

Operanden-Struktur für INTERFACE4

Die Operanden-Struktur von INTERFACE4 wird als ‘parameter_description’ bezeichnet und ist wie folgt definiert. Die Struktur wird mit der Header-Datei ‘natuser.h’ ausgeliefert.

struct parameter_description		
void*	address	Adresse der Parameterdaten, nicht ausgerichtet, realloc() und free() sind nicht zulässig
int	format	Felddatentyp: NCXR_TYPE_ALPHA, usw. (natuser.h)
int	length	Länge vor Dezimalpunkt (wenn zutreffend)
int	precision	Länge hinter Dezimalpunkt (wenn zutreffend)
int	byte_length	Länge des Feldes in Bytes; “int” Dimensionszahl (0 bis IF4_MAX_DIM)
int	dimensions	Anzahl Dimensionen (0 bis IF4_MAX_DIM)
int	length_all	Gesamtlänge des Arrays in Bytes
int	flags	Mehrere Flag-Bits, durch OR bitweise miteinander kombiniert, Bedeutung: IF4_FLG_PROTECTED — Parameter ist schreibgeschützt; IF4_FLG_DYNAMIC — Parameter ist dynamische Variable; IF4_FLG_NOT_CONTIGUOUS — Array-Elemente berühren sich nicht (es steht ein Leerzeichen zwischen ihnen); IF4_FLG_AIV — ist anwendungsunabhängige Variable;
int	occurrences(IF4_MAX_DIM)	Array-Ausprägungen in jeder Dimension
int	indexfactors(IF4_MAX_DIM)	Array-Index-Faktoren für jede Dimension
void*	dynp	reserviert für interne Zwecke

Die Adresse des Array-Elements (i, j, k) errechnet sich wie folgt (besonders dann, wenn die Array-Elemente sich nicht berühren):

$$\text{elementaddress} = \text{address} + i * \text{indexfactors}(0) + j * \text{indexfactors}(1) + k * \text{indexfactors}(2)$$

Wenn das Array weniger als 3 Dimensionen hat, sind die letzten Ausdrücke wegzulassen.

INTERFACE4 — Parameter-Zugriff

Eine Reihe von Funktionen steht für den Zugriff auf die Parameter zur Verfügung. Der Ablauf der Verarbeitung ist wie folgt.

Das 3GL-Programm wird über das CALL-Statement mit der Option INTERFACE4 aufgerufen, und die Parameter werden an das 3GL-Programm wie oben beschrieben übergeben. Das 3GL-Programm kann jetzt die exportierten Funktionen von Natural verwenden, um entweder die Parameterdaten selbst oder Informationen über die Parameter, wie Format, Länge, Array-Informationen, usw. einzulesen.

Die exportierten Funktionen dienen auch dazu, Parameterdaten zurückzugeben. Mit dieser Technik ist der Zugriff auf Parameter gewährleistet, um ein vom 3GL-Programm bewerkstelligtes Überschreiben des Speichers zu verhindern. (Natural-Daten sind sicher: ein Überschreiben des Speichers im Bereich der Daten des 3GL-Programms ist noch möglich).

Exportierte Funktionen

- Get parameter information (Parameter-Informationen holen)
- Get parameter data (Parameter-Daten holen)
- Write back operand data (Operanden-Daten zurückschreiben).

Get Parameter Information (Parameter-Informationen holen)

Diese Funktion wird vom 3GL-Programm verwendet, um alle erforderlichen Informationen zu Parametern zu erhalten. Diese Informationen werden in einer als ‘struct parameter_description’ bezeichneten, strukturierten Parameterbeschreibung zurückgegeben (siehe Seite 71).

Prototyp:

```
int ncxr_get_parm_info (int parmnum, void *parmhandle,
struct parameter_description *descr);
```

Parameter	Beschreibung
parmnum	Ordnungszahl des Parameters. Diese identifiziert den Parameter der Liste der übergebenen Parameter. Bereich: 0 ... numparm-1.
parmhandle	Zeiger zur internen Parameter-Struktur
descr	Adresse einer ‘struct parameter_description’
return	0 OK -1 fehlerhafte Parameter-Nummer -2 interner Fehler -7 Schnittstellen-Versionskonflikt

Get Parameter Data (Parameterdaten holen)

Diese Funktion wird vom 3GL-Programm verwendet, um die Daten von beliebigen Parametern zu holen. Natural identifiziert den Parameter über die vorgegebene Parameter-Nummer und schreibt die Parameterdaten in die gegebene Pufferadresse in der gegebenen Pufferlänge. Wenn die Parameterdaten länger als die gegebene Pufferlänge sind, schneidet Natural die Daten bis auf die gegebene Länge ab.

Das externe 3GL-Programm kann die Funktion ‘ncxr_get_parm_info’ dazu verwenden, um die Länge der Parameterdaten anzufordern. Es gibt zwei Funktionen zum Holen von Parameterdaten: ‘ncxr_get_parm’ holt den gesamten Parameter (auch wenn der Parameter ein Array ist), während ‘ncxr_get_parm_array’ das angegebene Array-Element holt.

Wenn vom 3GL-Programm für “buffer” kein Speicher der angegebenen Größe (dynamisch oder statisch) zugewiesen wird, sind die Ergebnisse der Operation nicht vorhersehbar. Natural überprüft dann nur die Pointer auf Gleichheit mit Null.

Wenn Daten bei Variablen des Typs I2/I4/F4/F8 (Pufferlänge ungleich Parameter-Gesamtlänge) abgeschnitten werden, sind die Ergebnisse vom Maschinentyp (‘little endian’ = höherwertiges Byte vorne / ‘big endian’ = höherwertiges Byte hinten) abhängig. In einigen Anwendungen muß der User Exit programmiert werden, um keine statischen Daten zu verwenden, so daß eine Rekursion möglich wird.

Protoypen:

```
int ncxr_get_parm (int parmnum, void *parmhandle, int buffer_length, void *buffer)
```

```
int ncxr_get_parm_array (int parmnum, void *parmhandle,  
int buffer_length, void *buffer, int *indexes)
```

Diese Funktion ist identisch mit ‘ncxr_get_parm’, außer daß die Indizes für jede Dimension angegeben werden können. Die Indizes für unbenutzte Dimensionen sollten als 0 angegeben werden.

Parameter	Beschreibung
parmnum	Ordnungszahl des Parameters. Diese identifiziert den Parameter der Liste der übergebenen Parameter. Bereich: 0 ... numparm-1.
parmhandle	Zeiger zur internen Parameter-Struktur.
buffer_length	Länge des Puffers, wohin die angeforderten Daten geschrieben werden müssen.
buffer	Adresse des Puffers, wohin die angeforderten Daten geschrieben werden müssen. Dieser Puffer sollte ausgerichtet werden, um einen leichten Zugriff auf I2/I4/F4/F8-Variablen zu ermöglichen.
indexes	Array mit Index-Informationen.
return	Ein Wert < 0 verweist auf einen Fehler beim Einlesen der Informationen. Ein Wert von -1 verweist auf eine fehlerhafte Parameter-Nummer. Ein Wert von -2 verweist auf einen internen Fehler. Ein Wert von -3 verweist darauf, daß die Daten abgeschnitten wurden. Ein Wert von -4 verweist darauf, daß die Daten kein Array sind. Ein Wert von -7 verweist auf einen Schnittstellen-Versionskonflikt. Ein Wert von -100 verweist darauf, daß der Index für Dimension 0 außerhalb des zulässigen Bereichs liegt. Ein Wert von -101 verweist darauf, daß der Index für Dimension 1 außerhalb des zulässigen Bereichs liegt. Ein Wert von -102 verweist darauf, daß der Index für Dimension 2 außerhalb des zulässigen Bereichs liegt. Ein Wert von 0 verweist auf eine erfolgreiche Operation. Ein Wert > 0 verweist auf eine erfolgreiche Operation; allerdings sind die Daten nur genau diese Anzahl Bytes lang (Puffer war länger als die Daten).

Write back Operand Data (Operanden-Daten zurückschreiben)

Diese Funktionen werden vom 3GL-Programm verwendet, um die Daten auf beliebige Parameter zurückzuschreiben. Natural identifiziert den Parameter über die gegebene Parameter-Nummer und schreibt die Parameterdaten von der gegebenen Pufferadresse in der gegebenen Pufferlänge auf die Parameterdaten.

Wenn die Parameterdaten kürzer als die gegebene Pufferlänge sind, werden die Daten bis auf die Länge der Parameterdaten abgeschnitten, d.h. der Rest des Puffers wird ignoriert. Wenn die Parameterdaten länger als die gegebene Pufferlänge sind, werden die Daten nur in der angegebenen Pufferlänge kopiert, die verbleibenden Parameter bleiben davon unberührt. Dies gilt gleichermaßen für Arrays. Bei dynamischen Variablen als Parameter wird der Parameter auf die angegebene Pufferlänge geändert.

Wenn Daten bei Variablen des Typs I2/I4/F4/F8 (Pufferlänge ungleich Parameter-Gesamtlänge) abgeschnitten werden, sind die Ergebnisse abhängig vom Maschinentyp ('little endian' = höherwertiges Byte vorne / 'big endian' = höherwertiges Byte hinten). In einigen Anwendungen muß der User Exit programmiert werden, um keine statischen Daten zu verwenden, so daß eine Rekursion möglich wird.

Prototypen:

```
int ncxr_put_parm (int parmnum, void *parmhandle, int buffer_length, void *buffer)
```

```
int ncxr_put_parm_array (int parmnum, void *parmhandle, int buffer_length, void *buffer,  
int *indexes);
```

Parameter	Beschreibung
parmnum	Ordnungszahl des Parameters. Diese identifiziert den Parameter der Liste der übergebenen Parameter. Bereich: 0 ... numparm-1.
parmhandle	Zeiger zur internen Parameter-Struktur.
buffer_length	Länge der Daten, die in die Adresse des Puffers zurück zu kopieren sind, von wo die Daten herkommen.
indexes	Index-Informationen.
return	<p>Ein Wert < 0 verweist auf einen Fehler beim Zurückkopieren der Informationen.</p> <p>Ein Wert von -1 verweist auf eine fehlerhafte Parameter-Nummer. Ein Wert von -2 verweist auf einen internen Fehler. Ein Wert von -3 verweist darauf, daß zu viele Daten angegeben wurden. Zurückkopieren erfolgte über die Parameterlänge. Ein Wert von -4 verweist darauf, daß der Parameter kein Array ist. Ein Wert von -5 verweist darauf, daß der Parameter geschützt ist (konstant oder AD=0). Ein Wert von -6 verweist darauf, daß die Länge der dynamischen Variable aufgrund einer 'out of memory'-Bedingung (Bedingung: Speicher nicht verfügbar) nicht geändert werden konnte. Ein Wert von -7 verweist auf einen Schnittstellen-Versionskonflikt. Ein Wert von -100 verweist darauf, daß der Index für Dimension 0 außerhalb des zulässigen Bereichs liegt. Ein Wert von -101 verweist darauf, daß der Index für Dimension 1 außerhalb des zulässigen Bereichs liegt. Ein Wert von -102 verweist darauf, daß der Index für Dimension 2 außerhalb des zulässigen Bereichs liegt. Ein Wert von 0 verweist auf eine erfolgreiche Operation. Ein Wert > 0 verweist auf eine erfolgreiche Operation, allerdings sind die Parameter genau diese Anzahl Bytes lang (Länge des Parameters > gegebene Länge).</p>

Alle Funktionsprototypen werden in der Datei 'natuser.h' deklariert.

Teil II: CALL unter OpenVMS, UNIX und Windows

- Return Code
- User Exits unter Windows
- User Exits unter OpenVMS
- User Exits unter UNIX.

Return Code

Um den Condition Code einer aufgerufenen Funktion zu erhalten, können Sie die Natural-Systemfunktion RET verwenden.

Beispiel:

```
...
RESET #RETURN(B4)
CALL 'PROG1'
IF RET ('PROG1') > #RETURN
    WRITE 'ERROR OCCURRED IN PROGRAM1'
END-IF
...
```

User Exits unter Windows

Unter Windows sind User Exits nötig, um auf externe Funktionen zugreifen zu können, die mit einem CALL-Statement aufgerufen werden. Die User Exits müssen in eine sogenannte “DLL” (dynamic link library) gestellt werden. Weitere Informationen zu den User Exits finden Sie in folgender Datei:

`%NATDIR%\%NATVERS%\samples\sysexuex\readme.txt`

User Exits unter OpenVMS

Unter OpenVMS sind User Exits nötig, um auf externe Funktionen zugreifen zu können, die mit einem CALL-Statement aufgerufen werden. Die User Exits müssen in ein sogenanntes “Shareable Image” gestellt werden. Weitere Informationen zu den User Exits finden Sie in folgender Datei:

NATSAMPLES:readme.txt

User Exits unter UNIX

- Schritt 1 — Jump Table definieren
- Schritt 2 — Externe Funktionen schreiben
- Schritt 3 — Kompilieren und Linken
- Erstellen einer Shared Library
- Benutzen der Shared Library
- Generieren eines Statischen Nukleus
- Beispielprogramme.

Unter UNIX sind User Exits nötig, um externe Funktionen aufrufen zu können und auf Betriebssystem-Schnittstellen zugreifen zu können, die für Natural nicht verfügbar sind.

Die User Exits können entweder in eine sogenannte “Shared Library” gestellt, d.h. dynamisch gelinkt, oder in eine Library gestellt werden, die statisch an den Natural-Nukleus gelinkt wird.

Wenn sie in eine Shared Library gestellt werden, ist es nicht nötig, bei jeder Änderung eines User Exits Natural neu zu linken, wodurch sich Entwicklung und Testen von User Exits sehr vereinfacht. Diese Möglichkeit besteht unter allen Betriebssystemen, die Shared Libraries unterstützen.

Unter allen Betriebssystemen ist es möglich, User Exits in eine Library zu stellen, die an den Natural-Nukleus gelinkt wird, d.h. die User Exits werden statisch mit dem vorgelinkten Natural-Objekt “natraw.o” gelinkt.

Ein User Exit wird Natural in drei Schritten hinzugefügt:

1. Es muß eine sogenannte “Jump Table” erstellt werden, die es Natural ermöglicht, eine Verbindung zwischen dem Namen der mit einem CALL-Statement aufgerufenen Funktion und der Adresse der Funktion herzustellen.
2. Die in der Jump Table angegebenen Funktionen müssen geschrieben werden.
3. Bei einem dynamischen Link muß die Shared Library, die die User Exits enthält, neu erzeugt werden.
Bei einem statischen Link müssen die Jump Table und die externen Funktionen mit dem vorgelinkten Natural-Nukleus zusammengelinkt werden, um einen ausführbaren Natural-Nukleus, der die externen Funktionen unterstützt, zu erhalten.

Schritt 1 — Jump Table definieren

Ein Muster einer Jump Table — “jumptab.c” — finden Sie im Verzeichnis:

\$NATDIR/\$NATVERS/samples/sysexuex

Schritt 2 — Externe Funktionen schreiben

Jede Funktion hat drei Parameter und gibt eine Ganzzahl (long integer) zurück. Der Prototyp einer Funktion sollte wie folgt aussehen:

```
NATFCT myadd (nparm, parmptr, parmdec)
```

```
WORD nparm;
BYTE **parmptr;
FINFO *parmdec;
```

nparm	16-Bit Kurzwert ohne Vorzeichen, der die Gesamtzahl der übergebenen Operanden (<i>operand2</i>) enthält.
parmptr	Array der Adressen der übergebenen Operanden.
parmdec	Array von Feldinformationen für jeden übergebenen Operanden.

Der Datentyp FINFO ist wie folgt definiert:

```
typedef struct {
    unsigned char    TypeVar;    /* type of variable          */
    unsigned char    pb2;        /* if type == ('D', 'N', 'P' or 'T') ==> */
                                /* total num of digits       */
                                /* else                       */
    union {
        unsigned char    pb[2];  /* if type == ('D', 'N', 'P' or 'T') ==> */
        unsigned short   lfield; /* pb[0] = #dig before.dec.point */
    } flen;                    /* pb[1] = #dig after.dec.point */
                                /* else                       */
                                /* lfield = length of field    */
} FINFO;
```

Als nächstes muß das Modul, das die externen Funktionen enthält, geschrieben werden. Eine Muster-Funktion — “mycadd.c” — finden Sie im Verzeichnis:

\$NATDIR/\$NATVERS/samples/sysexuex

Schritt 3 — Kompilieren und Linken

Die Datei “nuser.h”, die von dem Musterprogramm eingefügt wird, wird mit Natural ausgeliefert. Sie enthält die Deklarationen für die Datentypen BYTE, WORD und die FINFO-Struktur, d.h. die Beschreibung der internen Darstellung jedes übergebenen Parameters.

- Im Falle dynamisch gelinkter User Exits muß die Shared Library, die die User Exits enthält, neu erzeugt werden.
- Im Falle statisch gelinkter User Exits muß der Natural-Nukleus neu gelinkt werden.

Es ist dringend empfohlen, hierzu die von der Software AG zur Verfügung gestellten Makefiles zu verwenden, da sie bereits die erforderlichen Compiler- und Linker-Parameter enthalten. Sie finden die Muster-Makefiles im Verzeichnis:

\$NATDIR/\$NATVERS/samples/sysexuex

Weitere Informationen finden Sie in den folgenden Abschnitten sowie in den Erklärungen innerhalb der Makefiles selbst.

Erstellen einer Shared Library

1. Aus dem Beispiel-Verzeichnis, das in

\$NATDIR/\$NATVERS/samples/sysexuex

enthalten ist, kopieren Sie folgende Dateien in Ihr Arbeitsverzeichnis:

```
Makedyn  
jumptab.c  
ncuxinit.c
```

2. Kopieren Sie die C-Source-Dateien, die Ihre User Exits enthalten, in dasselbe Arbeitsverzeichnis.
3. Editieren Sie die Datei “jumptab.c”, um die Namen und Funktionszeiger für Ihre User Exits anzugeben. Hierzu fügen Sie in Sektion 2 die externen Deklarationen der User Exits und in Sektion 3 die Namen/Funktionsadressen-Paare der User Exits ein. Eventuell können Sie auch die betreffenden Sektionen aus Ihrer Vor-2.2-Version von “jumptab.c” kopieren.
4. Editieren Sie das Makefile wie folgt:

Geben Sie die Namen der Objektdateien, die die User Exits enthalten, in folgender Zeile an:

```
USEROBSJS =
```

Geben Sie den Namen der Shared Library in folgender Zeile an:

```
USERLIB =
```

Falls Sie private Header-Dateien benötigen, geben Sie die Verzeichnisse, in denen sie enthalten sind, in folgender Zeile an:

```
INCDIR =
```

5. Um alle nicht benötigten Dateien zu löschen, führen Sie folgendes Kommando aus:

make -f Makedyn clean

6. Um die Shared Library zu kompilieren und zu linkern, führen Sie folgendes Kommando aus:

make -f Makedyn lib

Benutzen der Shared Library

Setzen Sie die Umgebungsvariable NATUSER auf die Libraries, die Sie benutzen möchten.
Zum Beispiel:

```
setenv NATUSER $NATDIR/$NATVERS/bin/<library-name>
```

Sie müssen einen voll qualifizierten Pfadnamen für die Shared Library angeben.

Sie können mehrere Pfade angeben, wenn Sie sie mit einem Doppelpunkt (:) voneinander trennen (wie die UNIX-Variable PATH).

Beispiel:

Siehe Muster-User-Exit-Funktion in `$NATDIR/$NATVERS/samples/sysexuex`.

Anmerkung:

Die Libraries werden in der Reihenfolge abgesucht, in der sie in NATUSER angegeben sind. Wenn eine Funktion mit demselben Namen in zwei Libraries vorkommt, ruft Natural daher immer diejenige aus der Library auf, die zuerst in NATUSER angegeben ist.

Generieren eines statischen Nukleus

1. Aus dem Beispiel-Verzeichnis, das in **\$NATDIR/\$NATVERS/samples/sysexuex** enthalten ist, kopieren Sie folgende Dateien in Ihr Arbeitsverzeichnis:
makefile
jumptab.c
2. Kopieren Sie die C-Source-Dateien, die Ihre User Exits enthalten, in dasselbe Arbeitsverzeichnis.
3. Editieren Sie die Datei “jumptab.c”, um die Namen und Funktionszeiger für Ihre User Exits anzugeben. Hierzu fügen Sie in Sektion 2 die externen Deklarationen der User Exits und in Sektion 3 die Namen/Funktionsadressen-Paare der User Exits ein. Eventuell können Sie auch die betreffenden Sektionen aus Ihrer Vor-2.2-Version von “jumptab.c” kopieren.
4. Editieren Sie das Makefile wie folgt:
Geben Sie die Namen der Objektdateien, die die User Exits enthalten, in folgender Zeile an:
USEROBSJ =
Falls Sie private Header-Dateien benötigen, geben Sie die Verzeichnisse, in denen sie enthalten sind, in folgender Zeile an:
INCDIR =
5. Führen Sie das Kommando “make” aus, um Informationen über weitere Verarbeitungsoptionen zu erhalten.

Beispiel:

Siehe Muster-User-Exit-Funktion in **\$NATDIR/\$NATVERS/samples/sysexuex**.

Beispielprogramme:

Nach erfolgreichem Kompilieren und Linken können die externen Programme von einem Natural-Programm aus aufgerufen werden. Entsprechende Natural-Beispielprogramme finden Sie in der Library SYSEXUEX.

CALL FILE

Structured-Mode-Syntax

```
CALL FILE 'program-name' operand1 operand2
        statement...
END-FILE
```

Reporting-Mode-Syntax

```
CALL FILE 'program-name' operand1 operand2
        statement...
[LOOP]
```

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S A	A N P I F B D T L C	ja	ja
Operand2	S A G	A N P I F B D T L C	ja	ja

Funktion

Das Statement CALL FILE dient dazu, ein nicht in Natural geschriebenes Programm aufzurufen, das einen Datensatz von einer Nicht-Adabas-Datei liest und diesen Datensatz an das aufrufende Natural-Programm zur Verarbeitung übergibt.

Das Statement CALL FILE initiiert eine Verarbeitungsschleife, die mit einem ESCAPE- oder STOP-Statement beendet werden muß. Um die Schleife in Abhängigkeit von verschiedenen Bedingungen zu beenden, können Sie mehrere ESCAPE-Statements verwenden.

Einschränkung

Innerhalb einer CALL FILE-Schleife dürfen die Statements AT BREAK, AT START OF DATA und AT END OF DATA nicht verwendet werden.

Kontrollfeld (*operand1*)

Operand1 dient dazu, Kontrollinformationen zu liefern.

Datensatz-Bereich (*operand2*)

Operand2 definiert den Datensatz-Bereich.

Das Format des zu lesenden Datensatzes kann mit Felddefinitionseinträgen (oder FILLER *nX*), die hinter dem ersten Feld des Datensatzes stehen, beschrieben werden. Die Felder, die dazu dienen, das Format des Datensatzes zu definieren, brauchen im Natural-Programm nicht vorher definiert werden. Dadurch ist gewährleistet, daß Natural die Felder benachbarten Speicherplätzen zuordnet.

Beispiel

Aufrufendes Programm:

```

/* EXAMPLE 'CFIEX1': CALL FILE
/*****
DEFINE DATA LOCAL
1 #CONTROL (A3)
1 #RECORD
  2 #A (A10)
  2 #B (N3.2)
  2 #FILL1 (A3)
  2 #C (P3.1)
END-DEFINE
/*****
CALL FILE 'USER1' #CONTROL #RECORD
IF #CONTROL = 'END'
  ESCAPE BOTTOM
END-IF
END-FILE
/*****
/* ... PROCESS RECORD ...
/*****
END

```

Die Byte-Belegung des vom aufgerufenen Programm an das Natural-Programm übergebenen Datensatzes sieht folgendermaßen aus:

```

CONTROL      #A      #B  FILLER  #C
(A3)         (A10)   (N3.2) 3X  (P3.1)

  xxx xxxxxxxxxxxxxxx xxxxxx xxx xxx

```

Aufgerufenes COBOL-Programm:

```

ID DIVISION.
PROGRAM-ID. USER1.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT USRFILE ASSIGN UT-S-FILEUSR.
DATA DIVISION.
FILE SECTION.
FD  USRFILE RECORDING F LABEL RECORD OMITTED
   DATA RECORD DATA-IN.
01  DATA-IN          PIC X(80).
LINKAGE SECTION.
01  CONTROL-FIELD    PIC XXX.
01  RECORD-IN        PIC X(21).
PROCEDURE DIVISION USING CONTROL-FIELD RECORD-IN.
BEGIN.
    GO TO FILE-OPEN.
FILE-OPEN.
    OPEN INPUT USRFILE
    MOVE SPACES TO CONTROL-FIELD.
    ALTER BEGIN TO PROCEED TO FILE-READ.
FILE-READ.
    READ USRFILE INTO RECORD-IN
    AT END
    MOVE 'END' TO CONTROL-FIELD
    CLOSE USRFILE
    ALTER BEGIN TO PROCEED TO FILE-OPEN.
GOBACK.

```

CALL LOOP

Structured-Mode-Syntax

```
CALL LOOP operand1 [operand2] ...40
      statement...
END-LOOP
```

Reporting-Mode-Syntax

```
CALL LOOP operand1 [operand2] ...40
      statement...
[LOOP]
```

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	ja	nein
Operand2	C S A G	A N P I F B D T L C	ja	ja

Funktion

Das Statement CALL LOOP dient dazu, eine Verarbeitungsschleife zu generieren, die den Aufruf eines Nicht-Natural-Programms beinhaltet.

Im Gegensatz zum CALL-Statement erzeugt das CALL LOOP-Statement eine Verarbeitungsschleife, die dazu dient, das Nicht-Natural-Programm wiederholt aufzurufen. Zu der CALL-Verarbeitung siehe CALL-Statement.

Programmname (*operand1*)

Der Name des aufgerufenen Programms (*operand1*) kann entweder als Konstante angegeben werden oder — falls je nach Programmlogik verschiedene Programme aufgerufen werden sollen — als alphanumerische Variable mit Länge 1 bis 8. Ein Programmname muß linksbündig in der Variablen stehen.

Parameter (*operand2*)

Mit dem CALL LOOP-Statement können Sie bis zu 40 Parameter angeben. Der Aufbau der Parameterliste entspricht der für das CALL-Statement. In der Parameterliste verwendete Felder können schon vorher definiert werden oder erst im CALL LOOP-Statement selbst.

Beenden der Schleife

Die mit CALL LOOP initiierte Verarbeitungsschleife muß mit einem ESCAPE-Statement beendet werden.

Einschränkung

Innerhalb einer CALL LOOP-Schleife dürfen die Statements AT BREAK, AT START OF DATA und AT END OF DATA nicht verwendet werden.

Beispiel

```
DEFINE DATA LOCAL
1 PARAMETER1 (A10)
END-DEFINE
CALL LOOP 'ABC' PARAMETER1
  IF PARAMETER1 = 'END'
    ESCAPE BOTTOM
  END-IF
END-LOOP
END
```

CALLNAT

$$\text{CALLNAT } \textit{operand1} \left[\text{[USING]} \left\{ \textit{operand2} \left[\text{(AD} = \left\{ \begin{matrix} \text{M} \\ \text{O} \\ \text{A} \end{matrix} \right\} \right] \right\} \right\} \right] \dots$$

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	ja	nein
Operand2	C S A G	A N P I F B D T L C G O	ja	ja

Verwandte Statements

DEFINE DATA PARAMETER, FETCH, PERFORM.

Funktion

Das Statement CALLNAT dient dazu, ein Natural-*Subprogramm* zur Ausführung aufzurufen.

Wenn das CALLNAT-Statement ausgeführt wird, wird die Ausführung des aufrufenden Objekts (d.h. des Objekts, das das CALLNAT-Statement enthält) unterbrochen und das aufgerufene Subprogramm ausgeführt. Die Ausführung des Subprogramms dauert an, bis entweder sein END-Statement erreicht ist oder die Verarbeitung des Subprogramms durch die Ausführung eines ESCAPE ROUTINE-Statements gestoppt wird. In beiden Fällen wird dann die Verarbeitung des aufrufenden Objekts mit dem nächsten Statement nach dem CALLNAT-Statement fortgesetzt.

Anmerkung:

Ein Natural-Subprogramm kann nur über ein CALLNAT-Statement aufgerufen werden; es kann nicht selbständig ausgeführt werden.

Subprogramm-Name (*operand1*)

Als *operand1* geben Sie den Namen des Subprogramms an, das aufgerufen werden soll. Dieser kann entweder als 1 bis 8 Zeichen lange Konstante angegeben werden oder — falls je nach Programmlogik unterschiedliche Subprogramme aufgerufen werden sollen — als alphanumerische Variable mit Länge 1 bis 8.

Der Name des Subprogramms darf ein Und-Zeichen (&) enthalten; zur Laufzeit wird dieses Zeichen durch den aktuellen Wert der Systemvariablen *LANGUAGE ersetzt. Dadurch ist es beispielsweise möglich, je nachdem in welcher Sprache eine Eingabe gemacht wird, zur Verarbeitung der Eingabe unterschiedliche Subprogramme aufzurufen.

Parameter (*operand2*)

Falls Parameter an das Subprogramm übergeben werden, muß die Struktur der Parameterliste in einem DEFINE DATA PARAMETER-Statement definiert werden. Die mit dem CALLNAT-Statement angegebenen Parameter sind die einzigen Daten, die dem Subprogramm vom aufrufenden Objekt zur Verfügung stehen.

Standardmäßig erfolgt die Übergabe der Parameter “by reference”, d.h. die Daten werden über Adreß-Parameter übergeben, die Parameterwerte selbst werden nicht übertragen.

Es besteht aber auch die Möglichkeit, Parameter “by value” zu übergeben, d.h. die Parameterwerte selbst zu übergeben. Hierzu definieren Sie die betreffenden Felder im DEFINE DATA PARAMETER-Statement des Subprogramms mit der Option BY VALUE bzw. BY VALUE RESULT).

- Für die Parameterübergabe “by reference” gilt: Reihenfolge, Format und Länge der Parameter im aufrufenden Objekt müssen genau den Angaben im DEFINE DATA PARAMETER-Statement des Subprogramms entsprechen. Die Namen der Variablen im aufrufenden Objekt und im aufgerufenen Subprogramm können unterschiedlich sein.
- Für die Parameterübergabe “by value” gilt: die Reihenfolge der Parameter im aufrufenden Objekt muß der Reihenfolge im DEFINE DATA PARAMETER-Statement des Subprogramms entsprechen. Formate und Längen der Variablen im aufrufenden Objekt und im Subprogramm können unterschiedlich sein, müssen aber datenübertragungskompatibel (vgl. entsprechende Tabelle im *Natural Referenzhandbuch*) sein. Die Namen der Variablen im aufrufenden Objekt und im aufgerufenen Subprogramm können unterschiedlich sein.

Um Parameterwerte, die im Subprogramm verändert wurden, an das aufrufende Objekt zurückgeben zu können, müssen Sie die betreffenden Felder mit BY VALUE RESULT definieren.

Mit BY VALUE (ohne RESULT) ist es nicht möglich, veränderte Parameterwerte an das aufrufende Objekt zurückzugeben (unabhängig von der “AD”-Angabe; vgl. unten).

Anmerkung:

Intern wird bei BY VALUE eine Kopie der Parameterwerte erzeugt. Das Subprogramm greift auf diese Kopie zu und kann sie modifizieren, was aber keinen Einfluß auf die Originalparameterwerte im aufrufenden Objekt hat.

Bei BY VALUE RESULT wird ebenfalls eine Kopie erzeugt, aber nach Beendigung des Subprogramms überschreiben die (modifizierten) Werte der Kopie die Originalparameterwerte.

Für beide Arten der Parameterübergabe sind folgende Punkte zu beachten:

Wenn als *operand2* eine Gruppe angegeben wird, werden die einzelnen in der Gruppe enthaltenen Felder an das Subprogramm übergeben; d.h. für jedes dieser Felder muß in der Parameter Data Area des Subprogramms ein entsprechendes Feld definiert werden.

Eine Gruppe darf in der Parameter Data Area eines Subprogramms nur *innerhalb* eines REDEFINE-Blocks redefiniert werden.

Bei der Übergabe eines Arrays muß die Anzahl seiner Dimensionen und Ausprägungen in der Parameter Data Area des Subprogramms denen in der CALLNAT-Parameterliste entsprechen.

Anmerkung:

Wenn mehrere Ausprägungen eines Arrays, das als Teil einer indizierten Gruppe definiert ist, mit dem CALLNAT-Statement übergeben werden, dürfen die entsprechenden Felder in der Parameter Data Area des Subprogramms nicht redefiniert werden, da sonst die falschen Adressen übergeben werden.

AD=

Wenn *operand2* eine Variable ist, können Sie sie wie folgt kennzeichnen.

AD=O	nicht modifizierbar
AD=M	modifizierbar
AD=A	nur für Eingabe

Standardmäßig gilt AD=M.

Wenn *operand2* eine Konstante ist, kann AD nicht explizit angegeben werden. Für Konstanten gilt immer AD=O.

AD=M

Standardmäßig kann der übergebene Wert eines Parameters im Subprogramm geändert und der geänderte Wert an das aufrufende Objekt zurückgegeben werden, wo er den ursprünglichen Wert überschreibt.

Ausnahme: bei einem Feld, das in der Parameter Data Area des Subprogramms mit BY VALUE definiert ist, wird kein Wert zurückgegeben.

AD=O

Wenn Sie einen Parameter mit AD=O kennzeichnen, kann der übergebene Wert im Subprogramm geändert, aber nicht an das aufrufende Objekt zurückgegeben werden; d.h. im aufrufenden Objekt behält das Feld seinen ursprünglichen Wert.

Anmerkung:

Intern wird AD=O wie BY VALUE verarbeitet (vgl. Abschnitt “Parameter-data-definition” in der Beschreibung des DEFINE DATA-Statements).

AD=A

Wenn Sie einen Parameter mit AD=A kennzeichnen, wird sein Wert nicht *an* das Subprogramm übergeben, sondern er erhält einen Wert *von* dem Subprogramm. Dies kann bei Remote-Subprogrammen, die in einer Client/Server-Umgebung mittels Natural RPC ausgeführt werden, dazu beitragen, die Menge der übertragenen Daten zu reduzieren. Wenn ein Subprogramm lokal ausgeführt wird, werden “AD=A”-Felder auf Leerwerte zurückgesetzt, bevor das Subprogramm aufgerufen wird.

Bei einem Feld, das in der Parameter Data Area des Subprogramms mit BY VALUE definiert ist, kann das aufrufende Objekt keinen Wert erhalten. Hier bewirkt AD=A lediglich, daß das Feld vor dem Aufruf auf Leerwert zurückgesetzt wird.

nX

Diese Notation ist auf Großrechnern nicht verfügbar.

Mit der Notation *nX* können Sie spezifizieren, daß die nächsten *n* Parameter übersprungen werden sollen (Beispiel: 1X, um den nächsten Parameter zu überspringen, oder 3X, um die nächsten drei Parameter zu überspringen); d.h. daß für die nächsten *n* Parameter keine Werte an das Subprogramm übergeben werden.

Ein zu überspringender Parameter muß mit dem Schlüsselwort OPTIONAL im DEFINE DATA PARAMETER-Statement des Subprogramms definiert werden. OPTIONAL bedeutet, daß ein Wert vom aufrufenden Objekt an einen solchen Parameter übergeben werden kann — aber nicht muß.

Weitere Hinweise

Ein Subprogramm kann seinerseits andere Subprogramme aufrufen.

Ein Subprogramm kann nicht auf die vom aufrufenden Objekt verwendete Global Data Area zugreifen.

Wenn ein Subprogramm seinerseits eine Subroutine oder Helproutine aufruft, kann es eine eigene Global Data Area haben und diese mit der Subroutine/Helproutine teilen.

Übertragung von Parametern mit dynamischen Variablen

Dynamische Variablen können als Parameter an ein aufgerufenes Programmobjekt (CALLNAT, PERFORM) übergeben werden. “Call-by-reference” ist möglich, weil dynamische Variablen einen zusammenhängenden Wertebereich darstellen.

Bei “call-by-value” wird die Variablen-Definition des Aufrufenden als Ausgangs-Operand und die Parameter-Definition als Zieloperand zugewiesen. “Call-by-value result” bewirkt des weiteren eine Umkehrung der Zuordnung. Bei “call-by-reference” müssen beide Definitionen DYNAMIC sein. Wenn nur eine von ihnen DYNAMIC ist, wird ein Laufzeitfehler hervorgerufen. Bei “call-by-value (result)” sind alle Kombinationen möglich.

Die folgende Tabelle veranschaulicht die gültigen Kombinationen statisch und dynamisch definierter Variablen des Aufrufenden und statisch und dynamisch definierter Parameter hinsichtlich der Übertragung von Parametern.

Call by Reference

Operand2 vom Aufrufenden	Parameter-Definition	
	Statisch	Dynamisch
Statisch	ja	nein
Dynamisch	nein	ja

Die Formate der dynamischen Variablen A oder B müssen miteinander übereinstimmen.

Call by Value (Result)

Operand2 vom Aufrufenden	Parameter-Definition	
	Statisch	Dynamisch
Statisch	ja	ja
Dynamisch	ja	ja

Anmerkung:

Bei statischen/dynamischen oder dynamischen/statischen Definitionen kann es vorkommen, daß ein Wert nach den Datenübertragungsregeln der entsprechenden Zuweisungen abgeschnitten wird.

Beispiel 1

Aufrufendes Programm:

```

/* EXAMPLE 'CNTEX1': CALLNAT
/*****
/* MAIN PROGRAM 'MAINP1'
/*****
DEFINE DATA LOCAL
1 #FIELD1 (N6)
1 #FIELD2 (A20)
1 #FIELD3 (A10)
END-DEFINE
/*****
CALLNAT 'SUBP1' #FIELD1 (AD=M) #FIELD2 (AD=O) #FIELD3 'P4 TEXT'
/* ...
END

```

Aufgerufenes Subprogramm:

```

/* SUBPROGRAM 'SUBP1'
/*****
DEFINE DATA PARAMETER
1 #FIELDA (N6)
1 #FIELDB (A20)
1 #FIELD C (A10)
1 #FIELD D (A7)
END-DEFINE
/*****
/* ...
END

```

Beispiel 2

Aufrufendes Programm:

```
/* EXAMPLE 'CNTEX2': CALLNAT
/*****
/* MAIN PROGRAM 'MAINP2'
/*****
DEFINE DATA LOCAL
1 #ARRAY1 (A3/1:10,1:10)
END-DEFINE
CALLNAT 'SUBP2' #ARRAY1 (2:5,*)
/* ...
```

Aufgerufenes Subprogramm:

```
/* SUBPROGRAM 'SUBP2'
/*****
DEFINE DATA PARAMETER
1 #ARRAY (A3/1:4,1:10)
END-DEFINE
/*****
/* ...
END
```

CLOSE CONVERSATION

CLOSE CONVERSATION $\left\{ \begin{array}{l} \{operand1\}... \\ *CONVID \\ ALL \end{array} \right\}$

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S A	I	ja	nein

Verwandte Statements

DEFINE DATA CONTEXT, OPEN CONVERSATION.

Funktion

Das Statement CLOSE CONVERSATION wird im Zusammenhang mit Natural Remote Procedure Call (RPC) verwendet. Es ermöglicht dem Client, eine Conversation zu schließen. Sie können die aktuelle Conversation, eine andere offene Conversation oder alle offenen Conversations schließen.

Anmerkung:

Ein Logon in eine andere Library bewirkt kein automatisches Schließen von Conversations.

Zu schließende Conversation

Um eine bestimmte offene Conversation zu schließen, geben Sie ihre ID als *operand1* an. *Operand1* muß eine Variable mit Format/Länge I4 sein.

*CONVID

Um die aktuelle Conversation zu schließen, geben Sie *CONVID an. Die ID der aktuellen Conversation wird bestimmt durch den Wert der Systemvariablen *CONVID.

ALL

Um alle offenen Conversations zu schließen, geben Sie ALL an.

Weitere Informationen und Beispiele

Siehe Beschreibung von Natural RPC in Ihrer *Natural RPC Documentation*.

CLOSE DIALOG

Dieses Statement ist nur unter Windows und Windows NT verfügbar.

CLOSE DIALOG [USING] [DIALOG-ID] { <i>operand1</i> *DIALOG-ID }
--

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S	I	ja	nein

Verwandtes Statement

OPEN DIALOG.

Funktion

Dieses Statement dient dazu, einen Dialog dynamisch zu schließen.

Anmerkung:

Wenn ein Modal-Dialog ein "Child" in einer Hierarchie von Dialogen ist, sollte der Modal-Dialog nicht eines seiner "Parents" schließen, da dies die weitere Verarbeitung blockieren würde.

Zu schließender Dialog

operand1

Operand1 ist die Kennung (ID) des Dialogs, der geschlossen werden soll.

*DIALOG-ID

Um den aktuellen Dialog zu schließen, geben Sie *DIALOG-ID an.

Weitere Informationen und Beispiele

Siehe Kapitel **Event-Driven Programming Techniques** im *Natural User's Guide for Windows*.

CLOSE PC

Dieses Statement ist nur mit Natural Connection verfügbar. Es ist in der Natural Connection-Dokumentation beschrieben.

CLOSE PRINTER

CLOSE PRINTER { *(logical-printer-name)*
(printer-number) }

Verwandtes Statement

DEFINE PRINTER.

Funktion

Das Statement CLOSE PRINTER dient dazu, einen bestimmten Drucker zu schließen. Mit dem CLOSE PRINTER-Statement können Sie explizit in einem Programm angeben, daß ein Drucker geschlossen werden soll.

In folgenden Situationen wird ein Drucker automatisch geschlossen:

- wenn ein DEFINE PRINTER-Statement, in dem derselbe Drucker wieder definiert ist, ausgeführt wird
- bei Erreichen des Kommando-Modus.

Drucker

Mit *logical-printer-name* oder *printer-number* geben Sie den Drucker an, der geschlossen werden soll. Name und Nummer entsprechen den Angaben in dem DEFINE PRINTER-Statement, in dem Sie den betreffenden Drucker definiert haben. Für den *logical-printer-name* gelten die gleichen Namenskonventionen wie für Benutzervariablen (siehe **Allgemeine Informationen** im *Natural Referenzhandbuchs*). Die *printer-number* kann eine Zahl von 1 bis 31 sein.

Beispiel

```

* EXAMPLE 'CLPEX1': CLOSE PRINTER
*****
DEFINE DATA LOCAL
  1 EMP-VIEW VIEW OF EMPLOYEES
    2 PERSONNEL-ID
    2 NAME
    2 FIRST-NAME
    2 BIRTH
  1 I-NAME (A20)
END-DEFINE
*
DEFINE PRINTER (PRT01=1)
*
REPEAT
  INPUT 'SELECT PERSON' I-NAME
  IF I-NAME = ' '
    STOP
  END-IF
  FIND EMP-VIEW WITH NAME = I-NAME
    WRITE (PRT01) 'NAME           : ' NAME ', ' FIRST-NAME
      /          'PERSONNEL-ID : ' PERSONNEL-ID
      /          'BIRTH         : ' BIRTH (EM=YYYY-MM-DD)
  END-FIND
*
  CLOSE PRINTER (PRT01)
*
END-REPEAT
END

```

CLOSE WORK FILE

CLOSE WORK [**FILE**] *work-file-number*

Verwandte Statements

READ WORK FILE, WRITE WORK FILE.

Funktion

Das Statement CLOSE WORK FILE dient dazu, eine bestimmte Arbeitsdatei zu schließen. Es erlaubt Ihnen, in einem Programm explizit anzugeben, daß eine Arbeitsdatei geschlossen werden soll.

Eine Arbeitsdatei schließt sich auch automatisch, wenn der Kommando-Modus erreicht ist, oder wenn eine Dateiende-Bedingung bei der Ausführung eines READ WORK FILE-Statements auftritt.

Arbeitsdatei

Als *work-file-number* geben Sie die Nummer der Arbeitsdatei an (mit der sie für Natural definiert ist), die geschlossen werden soll.

Automatisches Schließen

Eine Arbeitsdatei wird automatisch geschlossen:

- bei Erreichen des Kommando-Modus
- wenn bei Ausführung eines READ WORK FILE-Statements eine “End-of-File”-Bedingung auftritt
- bevor ein DEFINE WORK FILE-Statement ausgeführt wird, das der betreffenden Arbeitsdateinummer ein anderes Dataset zuweist.

Beispiel

```
/* EXAMPLE 'CWFEEX1': CLOSE WORK FILE
DEFINE DATA LOCAL
01 W-DAT (A20)
01 REC-NUM (N3)
01 I (P3)
END-DEFINE
REPEAT
  READ WORK FILE 1 ONCE W-DAT /* READ MASTER RECORD
  AT END OF FILE
    ESCAPE BOTTOM
  END-ENDFILE
  INPUT 'PROCESSING FILE' W-DAT (AD=0)
    / 'ENTER RECORDNUMBER TO DISPLAY' REC-NUM
  IF REC-NUM = 0
    STOP
  END-IF
  FOR I = 1 TO REC-NUM
    READ WORK FILE 1 ONCE W-DAT
    AT END OF FILE
      WRITE 'RECORD-NUMBER TOO HIGH, LAST RECORD IS'
      ESCAPE BOTTOM
    END-ENDFILE
  END-FOR
  I := I - 1
  WRITE 'RECORD' I ':' W-DAT
  CLOSE WORK FILE 1
END-REPEAT
END
```

COMPOSE

Dieses Statement kann nur verwendet werden, wenn das Bürosystem Con-nect (Version 2 oder höher) installiert ist.

COMPOSE

[RESETTING–Klausel]
[MOVING–Klausel]
[ASSIGNING–Klausel]
[FORMATTING–Klausel]
[EXTRACTING–Klausel]

Wenn Sie mehr als eine Klausel (clause) angeben, werden diese in der oben abgebildeten Reihenfolge verarbeitet.

Funktion

Mit diesem Statement können Sie die Textformatierung durch Con-form (das von Con-nect verwendete Textformattersystem) direkt von einem Natural-Programm aus auslösen.

Der zu formatierende Text kann entweder mittels Variablen zur Verfügung gestellt werden oder aus einem Con-nect-Textblock (einem Dokument, das Con-form-Formatierbefehle enthält) abgerufen werden.

Die Inhalte von Natural-Variablen können als Variablen an Con-form übergeben werden und dynamisch in den formatierten Text eingefügt werden.

Die Werte einer Con-form-Variablen können auch vom Textformatierer wieder an das Natural-Programm übergeben werden.

Wenn die Con-form-Anweisungen ausgeführt worden sind (d.h. ein formatiertes Dokument erstellt wurde), wird die Ausgabe an eines der folgenden Ziele geleitet:

- einen Natural-Report
- ein Dokument in der Con-nect-Systemdatei
- Variablen in dem Natural-Programm, das das COMPOSE-Statement ausführt
- ein Nicht-Natural-Programm (nur auf Großrechnern).

Klauseln

Die **RESETTING**-Klausel dient dazu, Informationen aus dem Pufferbereich des Textformatierers zu löschen und Speicherplatz im **COMPOSE**-Puffer (der auf Großrechnern mit dem **CSIZE**-Parameter im **Natural**-Parametermodul zugewiesen wird) freizumachen.

Die **MOVING**-Klausel dient dazu, Textzeilen in den Pufferbereich des Textformatierers oder direkt an den Formatierer zu übertragen und formatierten Text aus dem Arbeitsbereich des Formatierers zu lesen.

Die **ASSIGNING**-Klausel dient dazu, Textvariablen Werte von **Natural**-Variablen zuzuweisen.

Die **FORMATTING**-Klausel dient dazu, aus Rohtext und Con-form-Befehlen einen fertig formatierten Text, mit korrektem Zeilen- und Seitenumbruch, zu erstellen.

Die **EXTRACTING**-Klausel dient dazu, **Natural**-Variablen die Werte von Textvariablen zuzuweisen.

Der Formatiervorgang

Der Formatiervorgang beginnt, wenn die **FORMATTING**-Klausel des **COMPOSE**-Statements ausgeführt wird (unter Umständen auch dann, wenn bei beabsichtigter Texteingabe durch die **MOVING**-Klausel noch keine solche Eingabe vorliegt).

Solange der Formatiervorgang aktiv ist, werden Texteingaben, die aus der Ausführung des **COMPOSE MOVING**-Statements resultieren, direkt in den Arbeitsbereich des Formatierers übertragen (und können in späteren Formatiervorgängen nicht wiederverwendet werden).

Ist der Formatiervorgang nicht aktiv, werden Texteingaben im **COMPOSE**-Puffer ("**DATAAREA**") zwischengespeichert. Dadurch ist es möglich, die Eingabe für mehrere Formatiervorgänge wiederzuverwenden.

Da der Formatierpuffer am Ende des **Natural**-Programms nicht gelöscht wird, brauchen die betreffenden **COMPOSE**-Statements nicht alle in demselben **Natural**-Programm zu stehen; sie können auch in mehreren nacheinander aufgerufenen Programmen stehen.

Die Ausführung einer **RESETTING**- oder **FORMATTING**-Klausel oder das Auftreten eines schwerwiegenden Formatierfehlers beenden den aktiven Formatierdurchlauf.

Das Ende der Eingabedaten wird durch die **LAST**-Subklausel der **MOVING**-Klausel bestimmt.

Wenn ein Con-nect-Dokument als Eingabequelle verwendet wird, wird davon ausgegangen, daß das Ende der Eingabedaten am Ende des Dokuments erreicht ist.

Anmerkung:

Es empfiehlt sich, die STATUS-Subklausel der FORMATTING- bzw. MOVING-Klausel zu verwenden, um sicherzustellen, daß sich der Formatiervorgang jeweils im richtigen Status für den betreffenden Verarbeitungsschritt befindet.

Dialog-Modus

Dialog-Modus-Verarbeitung ist eine Reihe von Interaktionen zwischen dem Benutzerprogramm und dem Formatierer, die während der Formatierung der Eingabe und der Erzeugung der Ausgabe stattfinden.

Der Dialog-Modus erlaubt es dem Benutzerprogramm, auf jeder Stufe der Eingabehierarchie dem Formatierer Rohtext als Eingabe zu liefern. Er akzeptiert auch formatierte Ausgaben direkt aus der aktuellen Programmumgebung.

Der Dialog entsteht dadurch, daß der Formatiervorgang in eine Reihe von Schritten unterteilt wird, von denen jeder einzelne durch ein eigenes COMPOSE-Statement aufgerufen wird.

Dialog-Modus für Eingabe

In den Dialog-Modus für Eingabe gelangt man, wenn DATAAREA die Quelle des Eingabetextes ist oder der Formatierbefehl “.TE ON” auftaucht und wenn der Con-form-Datenbereich keinen weiteren zu verarbeitenden Text mehr enthält. Dialog-Modus für Eingabe wird durch das Wort “TERM” in der ersten STATUS-Variablen angezeigt.

Das Benutzerprogramm sollte daraufhin die erforderlichen Eingaben machen, indem es die MOVING-Funktion eines anschließend verarbeiteten COMPOSE-Statements aufruft. Es kann die Terminaleingaben beenden, indem es die LAST-Option der MOVING-Funktion verwendet oder durch “.TE OFF”, falls der Dialog-Modus durch “.TE ON” begonnen wurde, und zwar als Text mittels der MOVING-Funktion. Das Ende der Formatierung wird durch “END” (bzw. “ENDX” im Falle eines Fehlers) in der ersten Statusvariablen angezeigt.

Dialog-Modus für Ausgabe

In den Dialog-Modus für Ausgabe gelangt man, wenn das Ausgabeziel TO VARIABLES ist. Der Formatierer gibt die Kontrolle an das Natural-Programm zurück, sobald die angegebenen Natural-Variablen gefüllt sind oder ein Seitenende erreicht ist (je nachdem, was zuerst der Fall ist). Dialog-Modus für Ausgabe wird durch STRG in der ersten STATUS-Variablen angezeigt.

Das Benutzerprogramm sollte daraufhin die soeben in die Natural-Variablen gestellte formatierte Ausgabe nehmen und einen weiteren Satz Natural-Variablen als Ausgabeziel in einem anschließend verarbeiteten COMPOSE MOVING-Statement bereitstellen. Das Ende der Formatierung wird durch “END” (bzw. “ENDX” im Falle eines Fehlers) angezeigt.

Anmerkung:

Wird Dialog-Modus verwendet (siehe INPUT- und OUTPUT-Subklauseln), erstreckt sich der Formatiervorgang in der Regel über mehrere Ausführungen eines COMPOSE-Statements.

Dialog-Modus für Ein- und Ausgabe

Der Dialog-Modus kann auch für kombinierte Ein- und Ausgabeverarbeitung verwendet werden. Wenn der Formatierer weitere Eingabedaten verlangt (angezeigt durch “TERM”) oder eine Ausgabe erzeugt hat (angezeigt durch “STRG”), muß das Natural-Programm daher entsprechend reagieren.

Wenn Sie im Dialog-Modus für kombinierte Ein- und Ausgabeverarbeitung sind, kann der Formatierer jeweils nur eine Eingabezeile zur Zeit annehmen. Lediglich im reinen Eingabe-Modus kann er mehrere Zeilen annehmen.

Ausführung von COMPOSE-Statements im Dialog-Modus

Wie bereits erwähnt, gelangt man in den Dialog-Modus über ein COMPOSE FORMATTING-Statement, das eine Reihe von COMPOSE MOVING-Ausführungen umfaßt. Beachten Sie dabei bitte folgendes:

- COMPOSE ASSIGNING- und COMPOSE EXTRACTING-Statements sind nur bei aktivem Dialog-Modus gültig.
- COMPOSE RESETTING und FORMATTING erzwingen die sofortige Beendigung aller Formatierung.

Nicht-Natural-Programme (nur auf Großrechnern)

In Abhängigkeit von den in der FORMATTING-Klausel angegebenen Parametern ist es möglich, Ein- und Ausgabe von Nicht-Natural-Programmen zu verarbeiten. Solche Programme werden über den gleichen Mechanismus, wie er im CALL-Statement verwendet wird, aufgerufen.

COMPOSE tauscht mit diesen Programmen Parameter aus, und zwar unter Verwendung von Standard-Linkage-Konventionen (dynamisches Laden ist in einer CICS-Umgebung nicht möglich).

Anmerkung:

Verarbeitung der Ein-/Ausgabe durch Nicht-Natural-Programme ist nur auf Großrechnern möglich; auf anderen Plattformen werden die entsprechenden Teile des COMPOSE-Statements ignoriert.

Je nach dem Status des Formatiervorgangs werden zwei oder drei Parameter zwischen dem Formatierer und den Nicht-Natural-Programmen übergeben:

<p>Parameter 1 (Format/Länge A1)</p>	<p>Funktionscode wird von Formatierer an Nicht-Natural-Programme übergeben. Mögliche Werte:</p> <p>I — Initiieren (Eingabe, Ausgabe) O — Dokument öffnen (Eingabe) R — Eine Zeile des Dokuments lesen (Eingabe) W — Eine Ausgabezeile schreiben (Ausgabe) C — Dokument schließen (Eingabe) T — Beenden (Eingabe, Ausgabe).</p>
<p>Parameter 2 (Format/Länge B1)</p>	<p>Response Code wird von Nicht-Natural-Programmen an Formatierer übergeben. Mögliche Werte:</p> <p>X'00' — Funktion erfolgreich ausgeführt. X'01' — Bei Funktion "O": Dokument nicht gefunden. Bei Funktion "R": Dokument-Ende erreicht. X'FF' — Funktion nicht beendet.</p>
<p>Parameter 3 (Format A1/256)</p>	<p>Bei Funktionen "O" und "W" werden diese Parameter vom Formatierer an Nicht-Natural-Programme übergeben; Parameter von Funktion "R" werden von Nicht-Natural-Programmen an den Formatierer übergeben.</p> <p>Bytes 1 – 2: Geben die Länge <i>n</i> dieses Parameters an. Bytes 3 – 4: Leer. Bytes 5 – <i>n</i>: Funktion "O": Dokumentname. Funktion "R": Vom Nicht-Natural-Programm gelesene Zeile. Funktion "W": Ausgabezeile vom Formatierer.</p> <p>Ist ein Seitenvorschub erforderlich, steht vor der Ausgabe ein "N", ansonsten eine "1".</p> <p>Angaben zur Hervorhebung von Text (z.B. Fettdruck oder Kursivschrift) werden ignoriert, wenn die Ausgabe an ein Nicht-Natural-Programm übergeben wird.</p>

RESETTING-Klausel

RESETTING [DATAAREA TEXTAREA MACROAREA ALL]

Mit dieser Klausel kann folgendes aus dem Textformatpufferbereich gelöscht werden:

- **DATAAREA** löscht alle aktiven Textvariablen.
- **TEXTAREA** löscht alle Texteingabedaten.
- **MACROAREA** löscht alle Textmakros.
- **ALL** löscht alle aktiven Textvariablen, Texteingabedaten und Textmakros.

Anmerkung:

Aus Kompatibilitätsgründen bezieht sich das Wort TEXTAREA auf die DATAAREA des Formatierers, so wie sie in der MOVING-Klausel verwendet wird.

MOVING-Klausel

Je nach Status des Dialog-Modus können Sie eine der folgenden Formen der MOVING-Klausel verwenden:

Syntax 1

```
MOVING [operand1]...37 [TO DATAAREA]
      [LAST][STATUS [TO] operand2[operand3[operand4 [operand5] ]]]]
```

Syntax 2

```
MOVING { operand1 [TO DATAAREA]
        LAST } [OUTPUT] TO VARIABLES operand6...20
      [STATUS [TO] operand2[operand3 [operand4 [operand5] ]]]]
```

Syntax 3

```
MOVING OUTPUT [TO VARIABLES] operand6...20
      [STATUS [TO] operand2[operand3 [operand4 [operand5] ]]]]
```

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S A	A N P	ja	nein
Operand2	S	A	ja	ja
Operand3	S	B	ja	ja
Operand4	S	B	ja	ja
Operand5	S	B	ja	ja
Operand6	S A	A	ja	nein

Operand2 muß mit Format/Länge A4 definiert sein. *Operand3*, *operand4* und *operand5* müssen mit Format/Länge B4 definiert sein.

Mit dieser Klausel können ein oder mehrere Textwerte in den Textformatierpufferbereich übertragen werden (Syntax 1). Dieser Bereich kann als Eingabequelle für Formatieroperationen verwendet werden. Wartet der Textformatierer gerade auf Eingaben (siehe **Dialog-Modus**, Seite 108), wird der Text direkt an ihn übergeben, ohne im Con-form-Textbereich zwischengespeichert zu werden (Syntax 1 und 2).

Die Source-Eingabe wird mit der LAST-Option beendet. Wartet der formatierte Text gerade darauf, ausgegeben zu werden (siehe **Dialog-Modus**, Seite 108), dann dient Syntax 3 der MOVING-Klausel dazu, die Kontrolle vom Natural-Programm wieder an den Formatierer zu übergeben. Zur Beschreibung der Statusvariablen siehe FORMATTING-Klausel.

Syntax 1

Syntax 1 der MOVING-Klausel gilt, wenn die Formatierung noch nicht begonnen hat oder der Formatierer im Dialog-Modus für Eingabe ist und auf Eingabedaten wartet (“TERM” in der ersten Statusvariablen).

Syntax 2

Syntax 2 der MOVING-Klausel gilt, wenn der Formatierer im Dialog-Modus für Ein- und Ausgabe ist und auf weitere Eingabedaten wartet (“TERM” in der ersten Statusvariablen). In diesem Modus akzeptiert der Formatierer jeweils nur eine Eingabezeile zur Zeit.

Der Ausführungskontext kann sich zwischen einer Reihe ausgeführter COMPOSE-Statements ändern. Daher müssen die Ausgabevariablen neu angegeben werden, selbst wenn der Formatierer auf Eingabedaten wartet.

Syntax 3

Syntax 3 der MOVING-Klausel gilt, wenn der Formatierer im Dialog-Modus für Ausgabe (und möglicherweise gleichzeitig für Eingabe) ist und die Ausgabe an das Natural-Programm übergibt (“STRG” in der ersten Statusvariablen).

ASSIGNING-Klausel

ASSIGNING [TEXTVARIABLE] {operand1 = operand2 } ,...¹⁹

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	ja	nein
Operand2	C S	A N P	ja	ja

Mit dieser Klausel werden Con-form-Textvariablen Werte zugewiesen. Anschließende Formatiervorgänge können sich auf diese Textvariablen beziehen.

Die Namen von Textvariablen sollten in Großbuchstaben angegeben werden.

FORMATTING-Klausel

FORMATTING

[
OUTPUT-Subklausel
INPUT-Subklausel
STATUS-Subklausel
PROFILE-Subklausel
MESSAGES-Subklausel
ERRORS-Subklausel
ENDING-Subklausel
STARTING-Subklausel
]...

Diese Klausel veranlaßt Con-form dazu, eine formatierte Ausgabe zu erzeugen.

Die Formatier-Optionen werden in einer oder mehreren Subklauseln (*subclauses*) angegeben. Wenn keine Subklauseln angegeben werden, nimmt Con-form die gültigen Standard-Formatier-Optionen. Die Statusvariable wird im Dialog-Modus verwendet.

OUTPUT-Subklausel

Das Ausgabemedium. Dies kann ein Natural-Report, ein Connect-Büro, eine oder mehrere Natural-Variablen (oder ein Array von Natural-Variablen) oder ein Nicht-Natural-Programm sein.

INPUT-Subklausel

Das Eingabemedium. Dies kann ein Connect-Dokument, die COMPOSE-“DATAAREA” (siehe MOVING-Klausel), die Umgebung der Natural-Programme, die die COMPOSE-Statements ausführen (siehe MOVING-Klausel), ein Nicht-Natural-Programm oder eine Mischung dieser vier sein.

STATUS-Subklausel

Der Status des Formatiervorgangs. Der Formatiervorgang kann die mehrfache Ausführung eines COMPOSE-Statements umfassen (im Dialog-Modus). Zum Beispiel wird die Eingabe von einem Natural-Programm in den Arbeitsbereich des Formatierers gestellt, und die Ausgabe wird vom Formatierer-Arbeitsbereich an die Umgebung eines Natural-Programms (d.h. eine oder mehrere Natural-Variablen) übergeben. Deshalb muß das Natural-Programm über den Status der Formatierung informiert werden. Folgende Variablen werden während der Formatierung an das Natural-Programm übergeben:

- **State** — Kann einen der folgenden Werte enthalten: TERM – wenn der Dialog-Modus für die Eingabe bereit ist; STRG – wenn der Dialog-Modus für die Ausgabe bereit ist; END – wenn die Formatierung erfolgreich beendet wurde; ENDX – wenn die Formatierung nicht erfolgreich beendet wurde.
- **Position** — Seiten- und Zeilennummer des gerade formatierten Dokuments. Diese Angaben werden in zwei separaten Variablen (Seitenposition und Zeilenposition) gehalten.
- **Amount of Output Data** — Die Anzahl formatierter Ausgabezeilen, die an das Natural-Programm übergeben werden. Der Formatierer benutzt diese Zahl als Zeiger auf die nächste zu füllende Ausgabevariable. Der Wert erhöht sich um “1”, bevor die Ausgabezeile ausgegeben wird. Ist der aktuelle Wert außerhalb des Bereichs, wird der Wert auf “1” gesetzt.

PROFILE-Subklausel

Der Textblock, der vor den Eingaben verarbeitet wird.

MESSAGES/ERRORS-Subklauseln

Steuert die Ausgabe von Warnmeldungen und statistischen Informationen und die Fehlerverarbeitung.

OUTPUT-Subklausel

OUTPUT	$\left\{ \begin{array}{l} \text{(rep)} \\ \text{SUPPRESSED} \\ \text{CALLING } \textit{operand1} \\ \text{TO VARIABLES [CONTROL } \textit{operand2 operand3}] \textit{ operand4...20} \\ \text{DOCUMENToption} \end{array} \right\}$
---------------	--

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	ja	nein
Operand2	C S	A	ja	nein
Operand3	C S	A	ja	nein
Operand4	S A	A	ja	nein

Mit dieser Subklausel können Sie den formatierten Con-form-Text an eine bestimmte Ausgabestelle schicken.

Wird diese Subklausel weggelassen, wird die Ausgabe standardmäßig an den Natural-Hauptdrucker geschickt.

OUTPUT (*rep*)

Wenn die Ausgabe an einen Drucker geschickt wird (d.h. wenn die Reportnummer nicht "0" ist) und ein Con-nect-Druckerprofil geladen wurde (mit der Con-nect-API-Funktion Z-DRIVER), werden die Optionen zur Hervorhebung von Teilen des formatierten Ausgabetexts über die Einstellungen in diesem Profil gesteuert.

Wenn ein Druckerprofil aktiv ist und keine logische Seitenvorschubsteuerung angegeben wurde, werden Seitenvorschübe über die entsprechenden internen Natural-Nukleusfunktionen eingefügt.

Weitere Texthervorhebungsoptionen, die im gerade aktiven Con-nect-Druckerprofil nicht definiert sind, werden ignoriert.

Anmerkung:

Bei der Ausführung eines COMPOSE RESETTING ALL-Statements oder eines COMPOSE FORMATTING-Statements, bei dem die Ausgabe nicht an einen Report geschickt wird, wird ein Druckerprofil aus dem Arbeitsbereich des Formatierers entladen.

Wird die Ausgabe an Report 0 geschickt oder ist kein Druckerprofil aktiv, übergibt Connect die Verantwortung für die Ausgabeverarbeitung an die Natural-Nukleus-Routinen. In diesem Falle werden nur die Texthervorhebungsoptionen Fettdruck, Unterstreichung und Kursivschrift berücksichtigt (gilt nur auf Großrechnern; auf anderen Plattformen werden diese Texthervorhebungsoptionen ignoriert).

Anmerkung:

Ein in einem Statement `DEFINE PRINTER (n) OUTPUT 'CONNECT'` referenzierter Report darf nicht als Ausgabeziel in einem `COMPOSE FORMATTING`-Statement angegeben werden.

OUTPUT SUPPRESSED

Mit dieser Option wird die Ausgabe unterdrückt.

OUTPUT CALLING

Siehe Abschnitt **Nicht-Natural-Programme** (Seite 109).

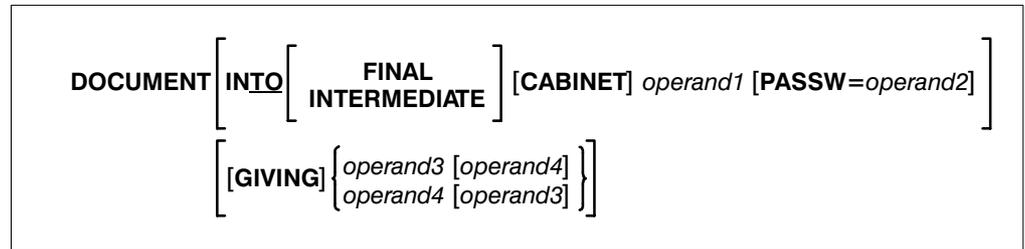
OUTPUT TO VARIABLES

Normalerweise wird der formatierte Text in seiner endgültigen Form an ein Array von Natural-Variablen übergeben. Jede Zeile füllt eine Variable (wenn eine Zeile nicht ganz in eine Variable paßt, wird sie abgeschnitten). Texthervorhebungsoptionen werden ignoriert, mit Ausnahme angegebener CONTROL-Variablen, die dazu dienen, Textteile hervorzuheben (d.h. fett oder unterstrichen auszugeben).

Wenn die CONTROL-Variablen “I” und “N” angegeben werden, wird formatierter Text in einem Zwischenformat erzeugt (d.h. durchsetzt mit logischen Steuersequenzen).

Operand2 und *operand3* müssen Format/Länge A1 haben.

Weitere Informationen, siehe Abschnitt **Dialog-Modus** (Seite 108).

DOCUMENT-option

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	ja	nein
Operand2	S	A	ja	nein
Operand3	S	B	ja	ja
Operand4	S	B	ja	ja

OUTPUT DOCUMENT

Operand3 (Format/Länge B10) wird vom Formatierer verwendet, um einen eindeutigen Schlüssel vom Dokument an das Natural-Programm zurückzugeben. Er wird nur aus Kompatibilitätsgründen unterstützt.

Operand4 (Format/Länge B4) wird vom Formatierer verwendet, um eine ISN, die auf ein formatiertes Ausgabedokument zeigt, an das Natural-Programm zurückzugeben. Diese ISN kann sinnvoll sein bei der Referenzierung eines Dokuments in aufeinanderfolgenden Aufrufen von Con-nect-APIs.

Wenn *operand1* (der bis zu 8 Zeichen lang sein darf) nicht angegeben wird, wird das Dokument in das aktuelle Benutzerbüro gestellt (d.h. das Büro, dessen ID der gerade aktive Natural-Benutzer entspricht).

Ein Paßwort (bis zu 8 Zeichen) muß angegeben werden, wenn ein Dokument in einem Büro gespeichert wird, auf das der aktuelle Benutzer keinen Zugriff hat.

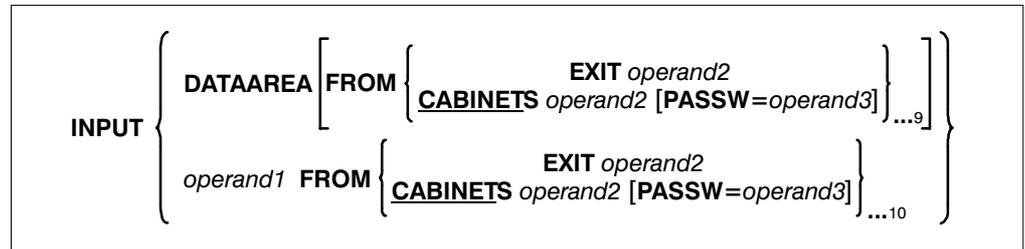
Con-form erzwingt, daß Con-nect-Zugriffsbeschränkungen eingehalten werden, und akzeptiert nur Büro-IDs, die in Con-nect definiert sind.

Anmerkung:

Büro-IDs müssen in Großbuchstaben angegeben werden.

Das Dokument wird ohne Namen in das Fach “COMPOSE” gestellt. In der Beschreibungszeile steht der Name des Programms, das das COMPOSE FORMATTING-Statement ausgeführt hat, sowie Datum und Uhrzeit der Ausführung.

Wenn Sie das Schlüsselwort INTERMEDIATE weglassen, wird das Dokument fertig formatiert erstellt. In diesem Falle werden Text hervorhebungsoptionen wie etwa Fettdruck oder Kursivschrift ignoriert.

INPUT-Subklausel

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	ja	nein
Operand2	C S	A	ja	nein
Operand3	S	A	ja	nein

Mit dieser Subklausel wird angegeben, woher die Eingaben für den Textformatierer kommen. Die Eingaben können aus dem Con-form-Datenbereich kommen (eine Mischung aus Text aus dem Datenbereich und aus dem Dialog-Modus ist ebenfalls möglich), der mit einer oder mehreren MOVING-Operationen gefüllt worden sein muß, oder mit einem Textblock (durch *operand1* angegeben).

Der Textblock kann einem Con-nect-Büro entstammen oder über ein Nicht-Natural-Programm (siehe Seite 109) zur Verfügung gestellt werden. Beim Aufrufen eines solchen Programms gelten die für das CALL-Statement gültigen Konventionen. Es kann eine Hierarchie von Con-nect-Büros oder Nicht-Natural-Programmen angegeben werden, von denen jede nacheinander nach dem in *operand1* benannten Textblock abgesehen wird.

Ein Paßwort muß angegeben werden, wenn ein Dokument in einem Büro gespeichert wird, auf das der aktuelle Benutzer keinen Zugriff hat.

Con-form erzwingt, daß Con-nect-Zugriffsbeschränkungen eingehalten werden, und akzeptiert nur Büro-IDs, die in Con-nect definiert sind.

Ohne diese Subklausel wird standardmäßig der Con-form-Datenbereich verarbeitet.

Anmerkung:

Büro- und Textblock-IDs müssen in Großbuchstaben angegeben werden.

STATUS-Subklausel

$\left[\text{STATUS } \textit{operand1} \left[\textit{operand2} \left[\textit{operand3} \left[\textit{operand4} \right] \right] \right] \right]$
--

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S	A	ja	nein
Operand2	S	B	ja	nein
Operand3	S	B	ja	nein
Operand4	S	B	ja	nein

Operand 1 enthält die Statusvariable “State”.

Operand 2 enthält die Statusvariable “Position (Seitennummer)”.

Operand 3 enthält die Statusvariable “Position (Zeilennummer)”.

Operand 4 enthält die Statusvariable “Amount of Output Data” (Ausgabedatenmenge).

PROFILE-Subklausel

PROFILE <i>operand1</i>

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	ja	nein

Diese Subklausel bewirkt, daß der Inhalt des angegebenen Textblocks verarbeitet wird, bevor in der *INPUT-Subklausel* angegebene Eingaben verarbeitet werden (standardmäßig wird ein Textblock nicht als Profil verarbeitet).

MESSAGES-Subklausel

MESSAGES { [LISTED] [ON] (<i>rep</i>) SUPPRESSED }
--

Bei Beendigung der Formatierung werden Warnmeldungen und statistische Informationen ausgegeben. SUPPRESSED bewirkt, daß keine Meldungen ausgegeben und Fehler ignoriert werden.

ERRORS-Subklausel

ERRORS { [LISTED] [ON] (<i>rep</i>) INTERCEPTED }

Mit dieser Subklausel können Sie bestimmen, was im Falle eines Formatierungsfehlers geschehen soll. Der Fehler kann entweder ignoriert, von einer Natural-Standard-Fehleroutine verarbeitet oder in einem bestimmten Natural-Report (*rep*) aufgelistet werden.

Anmerkung:

Fehler und Meldungen schließen einander aus. Bei manchen Fehlern wird die Natural-Fehleroutine aufgerufen, selbst wenn eine andere Option angegeben wurde.

Fehler und Meldungen dürfen nicht an einen Report geschickt werden, der über das Statement DEFINE PRINTER (n) OUTPUT 'CONNECT' an Con-nect geleitet wird.

ENDING-Subklausel

ENDING { [AT] [PAGE] <i>operand1</i> AFTER <i>operand1</i> [PAGES] }
--

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	N P	ja	nein

Diese Subklausel bewirkt, daß die Ausgabe des formatierten Textes nach einer bestimmten Seite (Seitennummer) unterdrückt wird bzw. die Ausgabe sich auf eine festgelegte Anzahl von Seiten beschränken soll.

STARTING-Subklausel

STARTING [FROM] [PAGE] operand1
--

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	N P	ja	nein

Diese Subklausel bewirkt, daß die Ausgabe des formatierten Textes solange unterdrückt wird, bis eine bestimmte Seitennummer (*operand1*) erreicht ist.

EXTRACTING-Klausel

EXTRACTING [TEXTVARIABLE] {operand1=operand2},...¹⁹

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S	A N P	ja	ja
Operand2	C S	A	ja	nein

Mit dieser Klausel können Sie Natural-Variablen die Werte von Textvariablen zuweisen. Die aktuellen Textvariablen-Werte können das Ergebnis eines vorhergehenden Formatiervorgangs sein.

Die Namen von Textvariablen müssen in Großbuchstaben angegeben werden.

Beispiel 1

```
COMPOSE RESETTING ALL
      FORMATTING INPUT 'TEXT' FROM CABINET 'TLIB'
      OUTPUT (1)
      MESSAGES LISTED ON (0)
```

Ergebnis des obigen COMPOSE-Statements: der formatierte Text des im Con-nect-Büro TLIB gespeicherten Textblocks TEXT wird auf Report 1 ausgegeben. Fehler und statistische Angaben werden auf Report 0 (dem Standarddrucker) ausgegeben.

Beispiel 2

```
COMPOSE RESETTING ALL
COMPOSE MOVING '.FI ON' 'This is an example'
COMPOSE MOVING 'for use of CON-FORM from'
      'within NATURAL applications' LAST
COMPOSE FORMATTING
```

Obige COMPOSE-Statements bewirken eine formatierte Textausgabe an Report 0 (Standarddrucker).

Beispiel 3

```
COMPOSE ASSIGNING 'VAR1' = 'Text1', 'VAR2' = 540
```

Obiges COMPOSE-Statement bewirkt die Zuordnung von Werten zu den Con-form-Textvariablen &VAR1 und &VAR2 in einer Con-nect-Prozedur.

Beispiel 4

Textblock “XYZ” in “XYLIB”:

```
.FI ON
Dear Mr &name.,
.II
I am pleased to invite you to a presentation of our new product &prod..
```

Natural-Programm:

```
...
INPUT #NAME (A32) #PROD (A32)
COMPOSE ASSIGNING 'NAME' = #NAME, 'PROD' = #PROD
      FORMATTING INPUT 'XYZ' FROM CABINET 'XYLIB'
      OUTPUT (1) MESSAGES SUPPRESSED
...
```

Vom Programm erzeugte Eingabe-Map:

```
#NAME Davenport
#PROD NATURAL 2.2
```

Erzeugte Ausgabe:

```
Dear Mr Davenport,

I am pleased to invite you to a presentation of our new product NATURAL 2.2.
```

Beispiel 5

Dies ist ein Beispiel für Formatierung im Dialog-Modus mit kombinierter Eingabe-/Ausgabe-Verarbeitung. Das Beispielprogramm initiiert Con-forms zeilenorientierten Formatiermodus, übergibt einige Kommandos/Variablen an Con-form und führt eine Subroutine aus, die Statusinformationen und die formatierte Ausgabezeile auf dem Schirm anzeigt.

```

DEFINE DATA LOCAL
01 #LINES_PER_PERFORM(P5) /* counts repeat-loops per PERFORM CNF_OUT
01 #TRACE(A1) INIT<'N'> /* if 'Y' displays additional trace-infos
01 #OUT_FORM(A1) INIT<'F'> /* output-format
01 #START_PAGE (P3) INIT<1> /* beginning of display
01 #CNTR (P5) /* Loop-Counter
01 #STATI /* Status-Information
02 #STATUS (A4) /* can be STRG TERM END or ENDX
02 #PAGE (B4) /* actual page-number
02 #LINE (B4) /* actual line-number on page (not .tt/.bt)
02 #NO_LINES (B4) /* number of lines returned
02 REDEFINE #NO_LINES
03 #NO_LINES_I (I4)
01 #OUTPUT(A30/4) /* output of formatted line
01 #INDEX (P3) /* index as pointer to out line
END-DEFINE
*
SET KEY ALL
SET CONTROL 'M9'
INPUT
008/008 'Demonstration of formatted output to Variable'(I)
/ 08X 'Enter page to start display : ' #START_PAGE(AD=MIL)
/ 08X 'Display additional trace-data ?:' #TRACE(AD=MIT)
/ 08X 'Please select the output-format:' #OUT_FORM(AD=MIT)
' (F=Final without BP/US-marks'
/ 44X 'M=Final with BP/US-marks "<>"'
/ 09X '(only, if CMF-Zap 2056 applied =>) I=Intermediate)'
/ 50X 'PF3=Exit'(I)
*
IF *PF-KEY EQ 'PF3'
SET CONTROL 'MB'
STOP
END-IF
*
IF NOT (#OUT_FORM EQ 'F' OR EQ 'M' OR EQ 'I')
REINPUT ' Please enter valid code!' MARK *#OUT_FORM ALARM
END-IF
*

```

```

WRITE TITLE LEFT
  'Stat * Page      * Line      * No.Lines >> Formatted Output'(I)
  / '-'(79)(I)
*
SET CONTROL 'MB'
COMPOSE RESETTING ALL /* clear all con-form buffers
RESET #NO_LINES
*
* start line-oriented formatting-mode here
* starting from 0
DECIDE ON FIRST VALUE OF #OUT_FORM
  VALUE 'F'
    COMPOSE FORMATTING
      OUTPUT TO VARIABLES #OUTPUT (1:4)      /* to Output
      STATUS #STATUS #PAGE #LINE #NO_LINES /* get Status
  VALUE 'M'
    COMPOSE FORMATTING
      OUTPUT TO VARIABLES CONTROL '<' '>'
      #OUTPUT (1:4)      /* to output
      STATUS #STATUS #PAGE #LINE #NO_LINES /* get Status
  VALUE 'I'
    COMPOSE FORMATTING
      OUTPUT TO VARIABLES CONTROL 'I' 'N'
      #OUTPUT (1:4)      /* to output
      STATUS #STATUS #PAGE #LINE #NO_LINES /* get Status
  NONE
  STOP
END-DECIDE
*
RESET #NO_LINES
*
* put some commands to con-form to see something
*
COMPOSE MOVING
  '.pl 16;.hs 2;.tt 1Formatierung in Variable//;.tt 2//' /* Cmd
  OUTPUT TO VARIABLES #OUTPUT (1:4)      /* to Output
  STATUS #STATUS #PAGE #LINE #NO_LINES /* get Status
*
COMPOSE MOVING
  '.fs 1;.bt Ende Seite #//;.fi on;.tb *=15' /* Commands
  OUTPUT TO VARIABLES #OUTPUT (1:4)      /* to Output
  STATUS #STATUS #PAGE #LINE #NO_LINES /* get Status
*
*
* loop 40-times, send commands to con-form and display output
*
```

```

COMPOSE ASSIGNING 'Wert' = '1-20' /* Assign value to variable &Wert
*
FOR #CNTR 1 40 /* Loop some time
  IF #STATUS NE 'TERM' /* no wait-for-input => error!!!!
    IF #STATUS EQ 'STRG'
      IGNORE
    ELSE
      WRITE 'Unexpected Status-code' #STATUS(AD=OI) 'found!'
        / 'Execution has stopped....'
      STOP
    END-IF
  END-IF
*
  IF #CNTR EQ 21
    COMPOSE ASSIGNING 'Wert' = '21-40' /* Assign a variable-value
  END-IF
  COMPOSE ASSIGNING 'CNTR' = #CNTR /* Again assignment
  COMPOSE MOVING
    '.BP;&Wert *Durchlauf &CNTR;.BR' /* Commands
    OUTPUT TO VARIABLES #OUTPUT (1:4) /* to Output
    STATUS #STATUS #PAGE #LINE #NO_LINES /* get Status
    PERFORM CNF-OUT /* show result
  END-FOR
  COMPOSE MOVING
    LAST /* End-Of-Processing
    OUTPUT TO VARIABLES #OUTPUT (1:4) /* to Output
    STATUS #STATUS #PAGE #LINE #NO_LINES /* get Status
*
  IF #TRACE EQ 'Y'
    WRITE 'End of processing...'(I)
  END-IF
*
* Subroutines
*
PERFORM CNF-OUT
*
* Subroutine to display any waiting output from con-form
*

```

```

DEFINE SUBROUTINE CNF-OUT
  RESET #LINES_PER_PERFORM
  REPEAT UNTIL #STATUS EQ 'TERM' /* TERM = input waiting
    PERFORM BREAK /* do some break-processing
    AT BREAK OF #PAGE
      IF #PAGE GT #START_PAGE
        WRITE '-'(79)(I)
      END-IF
      IF #TRACE EQ 'Y'
        WRITE 'End of this page...'(I)
      END-IF
      NEWPAGE
    END-BREAK
    IF #PAGE GE #START_PAGE /* show line of output
      IF #NO_LINES_I GT 0
        FOR #INDEX 1 #NO_LINES_I
          ADD 1 TO #LINES_PER_PERFORM /* count loops
          WRITE NOTIT NOHDR #STATUS '*' #PAGE '*' #LINE
            '*' #NO_LINES
            '>>' #OUTPUT (#INDEX)
        END-FOR
      END-IF
    END-IF
    IF #STATUS NE 'STRG' /* if no wait on out
      ESCAPE BOTTOM
    END-IF
    RESET #NO_LINES
    COMPOSE MOVING
      OUTPUT TO VARIABLES #OUTPUT (1:4) /* get Output
      STATUS #STATUS #PAGE #LINE #NO_LINES /* Status
    END-REPEAT
  *
  IF #TRACE EQ 'Y'
    WRITE 'Count of Lines per PERFORM was'(I) #LINES_PER_PERFORM(AD=OI)
  END-IF
  *
END-SUBROUTINE
SET CONTROL 'MB'
END

```

COMPRESS

COMPRESS [NUMERIC] [FULL]

{ *operand1* [(parameter)]
SUBSTRING (*operand1,operand3,operand4*) [(parameter)] }...

INTO { *operand2*
SUBSTRING (*operand2,operand5,operand6*) }

[**LEAVING SPACE**
LEAVING NO [SPACE]
WITH [ALL] [DELIMITERS] [*operand7*]]

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S A G N	A N P I F B D T G O	ja	nein
Operand2	S	A	ja	ja
Operand3	C S	N P I	ja	nein
Operand4	C S	N P I	ja	nein
Operand5	C S	N P I	ja	nein
Operand6	C S	N P I	ja	nein
Operand7	C S	A	ja	nein

Verwandte Statements

EXAMINE, SEPARATE.

Funktion

Das Statement COMPRESS dient dazu, den Inhalt zweier oder mehrerer Operanden in ein einziges alphanumerisches Feld zu übertragen.

Ausgangsfelder (*operand1*)

Mit *operand1* geben Sie die Felder an, deren Inhalt übertragen werden soll.

Anmerkung:

Wenn operand1 eine Zeitvariable (Format T) ist, wird nur die Zeitkomponente des Variableninhalts übertragen, aber nicht die Datumskomponente.

Zielfeld (*operand2*)

Mit *operand2* geben Sie das Feld an, das die Werte aus den Ausgangsfeldern aufnehmen soll.

Wenn Sie das COMPRESS-Statement ohne weitere Optionen verwenden oder LEAVING SPACE (gilt auch standardmäßig) angeben, so werden die Werte im Zielfeld jeweils durch ein Leerzeichen voneinander getrennt.

Wenn Sie LEAVING NO SPACE angeben, werden die Werte im Zielfeld weder durch ein Leerzeichen noch durch ein anderes Zeichen voneinander getrennt.

FULL

Ohne FULL werden vorangestellte Nullen (bei numerischen Feldern) und nachfolgende Leerzeichen (bei alphanumerischen Feldern) aus den Ausgangsfeldern entfernt, bevor die Werte übertragen werden. Enthält ein numerisches Ausgangsfeld lauter Nullen, wird eine “0” übertragen.

Mit FULL werden die Werte der Ausgangsfelder in ihrer tatsächlichen Länge — d.h. inklusive vorangestellter Nullen und nachfolgender Leerzeichen — ins Zielfeld übertragen.

Beispiele:

1. COMPRESS 'ABC ' 001 INTO #TARGET WITH DELIMITER '*'

Inhalt von #TARGET ist: **ABC*1**

2. COMPRESS **FULL** 'ABC ' 001 INTO #TARGET WITH DELIMITER '*'

Inhalt von #TARGET ist: **ABC *001**

NUMERIC

Diese Option bestimmt, wie Vorzeichen und Dezimalzeichen behandelt werden:

- Ohne NUMERIC werden Dezimalkommas und Vorzeichen bei numerischen Ausgangswerten unterdrückt, bevor die Werte in das Zielfeld übertragen werden.
- Mit NUMERIC werden Dezimalkommas und Vorzeichen aus numerischen Ausgangswerten ebenfalls mit in das Zielfeld übertragen.

Beispiele:

1. COMPRESS -123 1.23 INTO #TARGET WITH DELIMITER '**'

Inhalt von #TARGET ist: **123*123**

2. COMPRESS **NUMERIC** -123 1.23 INTO #TARGET WITH DELIMITER '**'

Inhalt von #TARGET ist: **-123*1.23**

parameter

Als *parameter* können Sie die Option "PM=I" oder den Session-Parameter DF angeben:

PM=I

Zur Unterstützung von Sprachen, deren Schreibrichtung von rechts nach links ist, können Sie die Option "PM=I" angeben, um den Wert von *operand1* invers (d.h. von rechts nach links) in *operand2* zu übertragen.

Zum Beispiel hätte als Ergebnis der folgenden Statements das Feld #B den Inhalt "ZYXABC":

```
MOVE 'XYZ' TO #A
COMPRESS #A (PM=I) 'ABC' INTO #B LEAVING NO SPACE
```

Nachfolgende Leerzeichen in *operand1* werden entfernt (außer wenn FULL angegeben ist), dann wird der Wert Zeichen für Zeichen umgedreht und anschließend in *operand2* übertragen.

DF

Wenn *operand1* eine Datumsvariable ist, können Sie den Session-Parameter DF als *parameter* für diese Variable angeben. Der Session-Parameter DF ist im *Natural Referenzhandbuch* beschrieben.

SUBSTRING

Wenn *operand1* alphanumerisches Format hat, können Sie die SUBSTRING-Option verwenden, um nur einen Teil des Ausgangsfeldes zu übertragen.

Sie können die SUBSTRING-Option auch in der INTO-Klausel verwenden, um die Ausgangswerte in einen bestimmten Teil des Zielfeldes zu übertragen.

Die Verwendung der SUBSTRING-Option in einem COMPRESS-Statement entspricht in beiden Fällen der in einem MOVE-Statement. Einzelheiten zur SUBSTRING-Option finden Sie beim MOVE-Statement.

WITH DELIMITER (*operand7*)

Möchten Sie, daß die Werte im Zielfeld jeweils durch ein bestimmtes Zeichen voneinander getrennt werden, dann verwenden Sie die DELIMITER-Option:

- Wenn Sie WITH DELIMITER *operand7* angeben, werden die Werte durch das mit *operand7* angegebene Zeichen voneinander getrennt. *Operand7* muß ein einzelnes Zeichen sein. Wenn *operand7* eine Variable ist, muß sie Format/Länge A1 haben.
- Wenn Sie WITH DELIMITERS ohne *operand7* angeben, werden die Werte durch das (mit dem Session-Parameter ID definierte) Input-Delimiter-Zeichen voneinander getrennt.

ALL

Ohne ALL werden im Zielfeld Delimiter-Zeichen nur zwischen tatsächlich übertragenen Werten gesetzt.

Mit ALL wird im Zielfeld auch für jeden (nicht übertragenen) Leerwert ein Delimiter-Zeichen gesetzt. Das heißt, die Anzahl der Delimiter-Zeichen im Zielfeld ist gleich der Anzahl der Ausgangsfelder minus 1. Dies kann z.B. sinnvoll sein, wenn der Inhalt des Zielfeldes mit einem SEPARATE-Statement anschließend wieder aufgeteilt werden soll.

Beispiele:

1. COMPRESS 'A' ' ' 'C' ' ' INTO #TARGET WITH DELIMITER '*'
Inhalt von #TARGET ist: **A*C**
 2. COMPRESS 'A' ' ' 'C' ' ' INTO #TARGET WITH **ALL** DELIMITERS '*'
Inhalt von #TARGET ist: **A**C***
-

Verarbeitung

Die COMPRESS-Operation wird beendet, sobald entweder alle Operanden übertragen sind oder das Zielfeld voll ist.

Ist das Zielfeld länger als alle übertragenen Werte zusammen, so werden die verbleibenden Stellen mit Leerzeichen gefüllt. Ist das Zielfeld kürzer, wird der Wert abgeschnitten.

Falls *operand2* eine DYNAMISCHE Variable ist, wird die COMPRESS-Operation beendet, wenn alle Ausgangs-Operanden verarbeitet worden sind. Es werden keine Zeichen abgeschnitten. Die Länge von *operand2* nach der COMPRESS-Operation entspricht dann der gemeinsamen Länge der Ausgangs-Operanden. Die aktuelle Länge einer DYNAMISCHEN Variable kann durch die Systemvariable *LENGTH bestimmt werden. Allgemeine Informationen zu DYNAMISCHEN Variablen entnehmen Sie Ihrer *Natural User's Documentation* oder Ihrem *Natural Benutzerhandbuch*.

Beispiel 1

```

/* EXAMPLE 'CMPEX1S:' COMPRESS (STRUCTURED MODE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 MIDDLE-I
1 #COMPRESSED-NAME (A20)
END-DEFINE
/*****
LIMIT 4
READ EMPLOY-VIEW BY NAME
  COMPRESS FIRST-NAME MIDDLE-I NAME INTO #COMPRESSED-NAME
  DISPLAY NOTITLE FIRST-NAME MIDDLE-I NAME 5X #COMPRESSED-NAME
END-READ
/*****
END

```

FIRST-NAME	MIDDLE-I	NAME	#COMPRESSED-NAME
KEPA		ABELLAN	KEPA ABELLAN
ROBERT	W	ACHIESON	ROBERT W ACHIESON
SIMONE		ADAM	SIMONE ADAM
TIMMIE	D	ADKINSON	TIMMIE D ADKINSON

Äquivalentes Reporting-Mode-Beispiel: siehe Programm CMPEX1R in Library SYSEXRM.

Beispiel 2

```

/* EXAMPLE 'CMPEX2': COMPRESS LEAVING NO SPACE
/*****
LIMIT 4
READ EMPLOYEES BY NAME
  COMPRESS CURR-CODE (1) SALARY (1) INTO #CCSALARY (A20)
    LEAVING NO SPACE
  DISPLAY NOTITLE NAME CURR-CODE (1) SALARY (1) 5X #CCSALARY
/*****
END

```

NAME	CURRENCY CODE	ANNUAL SALARY	#CCSALARY
ABELLAN	PTA	1450000	PTA1450000
ACHIESON	UKL	10500	UKL10500
ADAM	FRA	159980	FRA159980
ADKINSON	USD	36000	USD36000

Beispiel 3

```

/* EXAMPLE 'CMPEX3': COMPRESS WITH DELIMITER
/*****
LIMIT 4
READ EMPLOYEES BY NAME
  COMPRESS CURR-CODE (1) SALARY (1) INTO #CCSALARY (A20)
    WITH DELIMITER '*'
  DISPLAY NOTITLE NAME CURR-CODE (1) SALARY (1) 5X #CCSALARY
/*****
END

```

NAME	CURRENCY CODE	ANNUAL SALARY	#CCSALARY
ABELLAN	PTA	1450000	PTA*1450000
ACHIESON	UKL	10500	UKL*10500
ADAM	FRA	159980	FRA*159980
ADKINSON	USD	36000	USD*36000

COMPUTE

Structured-Mode-Syntax

$$\left\{ \left[\begin{array}{l} \text{COMPUTE} \\ \text{ASSIGN} \end{array} \right] \text{ [ROUNDED] } \left\{ \text{operand1}[:]= \dots \left\{ \begin{array}{l} \text{arithmetic-expression} \\ \text{operand2} \end{array} \right\} \right. \right. \\ \left. \left. \left\{ \text{operand1} := \dots \left\{ \begin{array}{l} \text{arithmetic-expression} \\ \text{operand2} \end{array} \right\} \right. \right. \right. \left. \right\}$$

Reporting-Mode-Syntax

$$\left[\left[\begin{array}{l} \text{COMPUTE} \\ \text{ASSIGN} \end{array} \right] \text{ [ROUNDED] } \right] \left\{ \text{operand1}[:]= \dots \left\{ \begin{array}{l} \text{arithmetic-expression} \\ \text{operand2} \end{array} \right\} \right.$$

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S A M	A N P I F B D T L C G O	ja	ja
Operand2	C S A N	A N P I F B D T L C G O	ja	nein

Verwandte Statements

ADD, SUBTRACT, MULTIPLY, DIVIDE, MOVE.

Funktion

Das Statement COMPUTE dient zur Ausführung einer arithmetischen Operation sowie dazu, einem oder mehreren Feldern einen Wert zuzuweisen.

Sie können das Statement in Kurzform angeben und den Statement-Namen COMPUTE (bzw. ASSIGN) weglassen. Wenn Sie im *Structured Mode* den Statement-Namen weglassen, müssen Sie vor das Gleichheitszeichen (=) einen Doppelpunkt (:) schreiben. Wenn Sie die ROUNDED-Option verwenden, müssen Sie den Statement-Namen angeben.

Zu arithmetischen Operationen mit Arrays siehe auch den Abschnitt **Arithmetische Operationen mit Arrays** im *Natural Referenzhandbuch*.

Ergebnisfeld (*operand1*)

Operand1 nimmt das Ergebnis der arithmetischen Operation bzw. Zuweisung auf.

Zur Genauigkeit des Ergebnisses siehe Abschnitt **Arithmetische Operationen** im *Natural Referenzhandbuch*.

Ist *operand1* ein Datenbankfeld, ändert sich der Wert des Feldes auf der Datenbank dadurch nicht.

Falls *operand1* eine DYNAMISCHE Variable ist, wird er bis zur Länge von *operand2* oder bis zur Länge des Ergebnisses der *arithmetic-operation* aufgefüllt, und die Länge von *operand1* wird dann dementsprechend angepaßt. Die aktuelle Länge einer DYNAMISCHEN Variable kann durch die Systemvariable *LENGTH bestimmt werden. Allgemeine Informationen zu DYNAMISCHEN Variablen entnehmen Sie Ihrer *Natural User's Documentation* oder Ihrem *Natural Benutzerhandbuch*.

ROUNDED

Wenn Sie das Schlüsselwort **ROUNDED** angeben, wird der Wert auf- bzw. abgerundet, bevor er *operand1* zugewiesen wird. Die für das Runden gültigen Regeln finden Sie im Abschnitt **Arithmetische Operationen** im *Natural Referenzhandbuch*.

arithmetic-expression

Ein arithmetischer Ausdruck (*arithmetic-expression*) besteht aus einer oder mehreren Konstanten, Datenbankfeldern bzw. Benutzervariablen.

Mathematische Natural-Funktionen (siehe dazu das Kapitel **Systemfunktionen** im *Natural Referenzhandbuch*) können ebenfalls als arithmetische Operanden verwendet werden.

Die in einem arithmetischen Ausdruck verwendeten Operanden müssen eines der folgenden Formate haben: N, P, I, F, D oder T.

Zum Format der Operanden siehe auch den Abschnitt **Formatwahl im Hinblick auf die Verarbeitungszeit** im *Natural Referenzhandbuch*.

Die folgenden Operatoren können verwendet werden:

- () Klammern
- ** Potenzierung
- * Multiplikation
- / Division
- + Addition
- Subtraktion

Jedem Operatorzeichen sollte jeweils mindestens ein Leerzeichen vor- und nachgestellt werden, damit es nicht zu Konflikten mit Variablennamen, die eines dieser Zeichen enthalten, kommen kann.

Bei der Verarbeitung arithmetischer Operationen gilt folgende Reihenfolge:

- ① Klammerrechnung
- ② Potenzrechnung
- ③ Multiplikation/Division (von links nach rechts)
- ④ Addition/Subtraktion (von links nach rechts)

Ergebnisgenauigkeit einer Division

Die Genauigkeit (Anzahl der Dezimalstellen) des Ergebnisses einer Division in einem COMPUTE-Statement bestimmt sich entweder aus der Genauigkeit des ersten Operanden (Dividenten) oder der des ersten Ergebnisfeldes, je nachdem welche größer ist.

Bei einer Division von Ganzzahlen gilt dagegen folgendes: Die Ergebnisgenauigkeit einer Division von zwei *Ganzzahl-Konstanten* bestimmt sich aus der Genauigkeit des ersten Ergebnisfeldes; ist jedoch eine der beiden Ganzzahlen eine *Variable*, dann ist auch das Ergebnis eine Ganzzahl (d.h. ohne Dezimalstellen, ganz gleich welche Genauigkeit das Ergebnisfeld hat).

SUBSTRING

Wenn die Operanden alphanumerisches Format haben, können Sie die SUBSTRING-Option verwenden (in der gleichen Weise wie beim MOVE-Statement beschrieben), um *operand1* einen Teil von *operand2* zuzuweisen. Einzelheiten zur SUBSTRING-Option finden Sie beim MOVE-Statement.

Beispiel 1

```

/* EXAMPLE 'ASGEX1S': ASSIGN (STRUCTURED MODE)
DEFINE DATA LOCAL
  1 #A (N3)
  1 #B (A6)
  1 #C (N0.3)
  1 #D (N0.5)
  1 #E (N1.3)
  1 #F (N5)
  1 #G (A25)
  1 #H (A3/1:3)
END-DEFINE
/*****
ASSIGN #A = 5                               WRITE NOTITLE '=' #A
ASSIGN #B = 'ABC'                           WRITE '=' #B
ASSIGN #C = .45                             WRITE '=' #C
ASSIGN #D = #E = -0.12345                   WRITE '=' #D / '=' #E
ASSIGN ROUNDED #F = 199.999                 WRITE '=' #F
#G      := 'HELLO'                          WRITE '=' #G
#H (1) := 'UVW'
#H (3) := 'XYZ'                             WRITE '=' #H (1:3)
END

```

```

#A:      5
#B:     ABC
#C:     .450
#D:    -.12345
#E:    -0.123
#F:     200
#G:    HELLO
#H:    UVW      XYZ

```

Äquivalentes Reporting-Mode-Beispiel: siehe Programm ASGEX1R in Library SYSEXRM.

Beispiel 2

```

/* EXAMPLE 'CPTEX1S': COMPUTE (STRUCTURED MODE)
/*****
DEFINE DATA LOCAL
1 #I (P2)
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 SALARY (1:2)
1 #A (P4)
1 #B (N3.4)
1 #C (N3.4)
1 #CUM-SALARY (P10)
END-DEFINE
/*****
COMPUTE #A = 3 * 2 + 4 / 2 - 1
WRITE NOTITLE 'COMPUTE #A = 3 * 2 + 4 / 2 - 1' 10X '=' #A
/*****
COMPUTE ROUNDED #B = 3 - 4 / 2 * .89
WRITE 'COMPUTE ROUNDED #B = 3 -4 / 2 * .89' 5X '=' #B
/*****
COMPUTE #C = SQRT (#B)
WRITE 'COMPUTE #C = SQRT (#B)' 18X '=' #C
/*****
LIMIT 1
READ EMPLOY-VIEW BY PERSONNEL-ID STARTING FROM '20017000'
WRITE / 'CURRENT SALARY:' 4X SALARY (1)
  / 'PREVIOUS SALARY:' 4X SALARY (2)
FOR #I = 1 TO 2
  COMPUTE #CUM-SALARY = #CUM-SALARY + SALARY (#I)
END-FOR
WRITE 'CUMULATIVE SALARY:' #CUM-SALARY
END-READ
/*****
END

```

COMPUTE #A = 3 * 2 + 4 / 2 - 1	#A:	7
COMPUTE ROUNDED #B = 3 -4 / 2 * .89	#B:	1.2200
COMPUTE #C = SQRT (#B)	#C:	1.1045
CURRENT SALARY:		34000
PREVIOUS SALARY:		32300
CUMULATIVE SALARY:		66300

Äquivalentes Reporting-Mode-Beispiel: siehe Programm CPTEX1R in Library SYSEXRM.

CREATE OBJECT

```
CREATE OBJECT operand1 OF [CLASS] operand2
              [ON [NODE] operand3]
              [GIVING operand4]
```

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S		nein	nein
Operand2	C S	A	ja	nein
Operand3	C S	A	ja	nein
Operand4	S N	I	ja	nein

Funktion

Das Statement CREATE OBJECT dient zum Erstellen einer Instanz einer Klasse. Wenn ein CREATE OBJECT-Statement ausgeführt wird, überprüft NaturalX, ob der Name der im Statement angegebenen Klasse registriert ist. Ist dies der Fall, erstellt NaturalX das Objekt mittels DCOM. Ist dies nicht der Fall, sucht NaturalX nach einer Klasse mit diesem Namen in der aktuellen Natural-Library oder in den Steplibs und erstellt das Objekt lokal.

Objekt-Handle — *operand1*

Operand1 muß als Objekt-Handle (HANDLE OF OBJECT) definiert sein.

Die Objekt-Handle wird gefüllt, wenn das Objekt erfolgreich erstellt wurde. Wenn *operand1* nicht erfolgreich zurückgegeben wird, enthält er den Wert NULL-HANDLE.

Class-Name — *operand2*

Operand2 ist der Name der Klasse, für die das Objekt erstellt werden soll. Bei Klassen, die nicht registriert sind, muß er den im DEFINE CLASS-Statement definierten Klassen-Namen enthalten. Bei Klassen, die registriert sind, muß er entweder die ProgID der Klasse oder die Klasse GUID enthalten. Bei Natural-Klassen entspricht die ProgID dem im DEFINE CLASS-Statement angegebenen Klassen-Namen. Weitere Informationen entnehmen Sie dem Unterabschnitt **Registration with Natural** im Abschnitt **Distributing Object-based Natural Applications** der *NaturalX Documentation*.

```
CREATE OBJECT #01 OF CLASS "EMPLOYEE" or  
CREATE OBJECT #01 OF CLASS "653BCFE0-84DA-11D0-BEB3-10005A66D231"
```

Node — *operand3*

Als *operand3* geben Sie den Knoten an, auf dem das Objekt erstellt wird. Dies ist nur möglich, wenn die Klasse registriert wird.

Wenn die NODE-Klausel angegeben wird, dann wird der Versuch unternommen, das Objekt auf diesem Knoten zu erstellen. Wenn die NODE-Klausel nicht angegeben wird oder einen Leerwert enthält, wird das Objekt auf dem Knoten erstellt, der im System-Register unter dem Schlüssel "RemoteServerName" für diese Klasse angegeben ist. Wenn dieser Register-Schlüssel nicht angegeben wird, wird das Objekt in der lokalen Natural-Session erstellt. Zum Beispiel:

```
CREATE OBJECT #01 OF CLASS "Employee" ON NODE "volcano.iceland.com"
```

GIVING — *operand4*

Wenn die GIVING-Klausel angegeben wird, enthält *operand4* entweder die Natural-Meldungsnummer, falls ein Fehler auftritt, oder Null bei fehlerfreier Ausführung.

Wenn die GIVING-Klausel nicht angegeben wird, wird die Natural-Laufzeitfehlerverarbeitung ausgelöst, falls ein Fehler auftritt.

DECIDE FOR

```
DECIDE FOR { FIRST  
            EVERY } CONDITION  
            { WHEN logical-condition statement... } ...  
            [ WHEN ANY statement... ]  
            [ WHEN ALL statement... ]  
            WHEN NONE statement...  
END-DECIDE
```

Verwandte Statements

DECIDE ON, IF.

Funktion

Das Statement DECIDE FOR dient dazu, in Abhängigkeit von mehreren Bedingungen eine oder mehrere Handlungen auszuführen.

Anmerkung:

*Falls unter einer bestimmten Bedingung **keine** Handlung ausgeführt werden soll, geben Sie das Statement IGNORE in der betreffenden Klausel des DECIDE FOR-Statements an.*

FIRST/EVERY

Mit einem dieser Schlüsselwörter geben Sie an, ob nur die *erste* erfüllte Bedingung (FIRST) oder *alle* erfüllten Bedingungen (EVERY) verarbeitet werden sollen.

WHEN *logical-condition*

Mit dieser Klausel geben Sie eine oder mehrere logische Bedingungen (*logical-condition*) an, die verarbeitet werden sollen (siehe Abschnitt **Logische Bedingungen** im *Natural Referenzhandbuch*).

WHEN ANY

Mit WHEN ANY können Sie die *statements* angeben, die ausgeführt werden sollen, wenn *irgendeine* der angegebenen Bedingungen erfüllt ist.

WHEN ALL

Mit WHEN ALL können Sie die *statements* angeben, die ausgeführt werden sollen, wenn *alle* angegebenen Bedingungen erfüllt sind. Diese Klausel kann nur in Verbindung mit dem Schlüsselwort EVERY eingesetzt werden.

WHEN NONE

Mit WHEN NONE können Sie die *statements* angeben, die ausgeführt werden sollen, wenn *keine* der angegebenen Bedingungen erfüllt ist.

Beispiel 1

```

/* EXAMPLE 'DECEX1:' DECIDE FOR (USING FIRST OPTION)
/*****
/* IF FUNCTION = A AND PARM = X
/*   ROUTINE-A IS TO BE EXECUTED.
/* IF FUNCTION = B AND PARM = X
/*   ROUTINE-B IS TO BE EXECUTED.
/* IF FUNCTION = C THRU D
/*   ROUTINE-CD IS TO BE EXECUTED.
/* FOR ALL OTHER CASES,
/*   REINPUT STATEMENT IS TO BE EXECUTED.
/*****
DEFINE DATA LOCAL
1 #FUNCTION (A1)
1 #PARM (A1)
END-DEFINE
/*****
INPUT #FUNCTION #PARM
/*****
DECIDE FOR FIRST CONDITION
    WHEN #FUNCTION = 'A' AND #PARM = 'X'
        PERFORM ROUTINE-A
    WHEN #FUNCTION = 'B' AND #PARM = 'X'
        PERFORM ROUTINE-B
    WHEN #FUNCTION = 'C' THRU 'D'
        PERFORM ROUTINE-CD
    WHEN NONE
        REINPUT 'PLEASE ENTER A VALID FUNCTION'
        MARK **FUNCTION
END-DECIDE
/*****
END

```

```
#FUNCTION A #PARM Y
```

```
PLEASE ENTER A VALID FUNCTION
#FUNCTION A #PARM Y
```

Beispiel 2

```

/* EXAMPLE 'DECEX1E:' DECIDE FOR (EVERY OPTION)
/*****
DEFINE DATA LOCAL
1 #FIELD1 (N5.4)
END-DEFINE
/*****
INPUT #FIELD1
/*****
DECIDE FOR EVERY CONDITION
    WHEN #FIELD1 >= 0
        WRITE '#FIELD1 is positive or zero.'
    WHEN #FIELD1 <= 0
        WRITE '#FIELD1 is negative or zero.'
    WHEN FRAC(#FIELD1) = 0
        WRITE '#FIELD1 has no decimal digits.'
    WHEN ANY
        WRITE 'Any of the above conditions is true.'
    WHEN ALL
        WRITE '#FIELD1 is zero.'
    WHEN NONE
        IGNORE
END-DECIDE
/*****
END

```

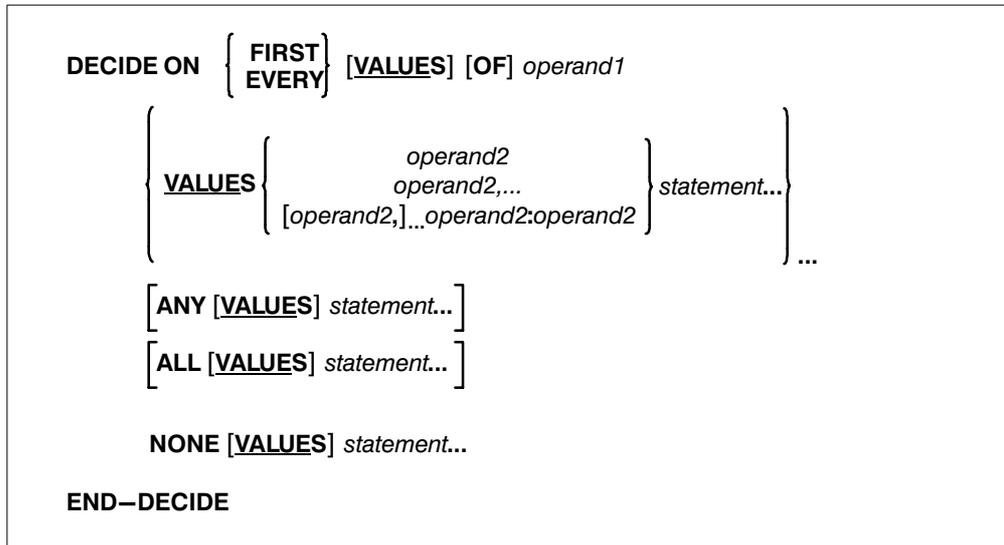
```
#FIELD1 42
```

```
Page      1
```

```
90-10-29 12:24:33
```

```
#FIELD1 is positive or zero.
#FIELD1 has no decimal digits.
Any of the above conditions is true.
```

DECIDE ON



Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S A N	A N P I F B D T L G O	ja	nein
Operand2	C S A	A N P I F B D T L G O	ja	nein

Verwandte Statements

DECIDE FOR, IF.

Funktion

Das Statement DECIDE ON dient dazu, in Abhängigkeit vom Wert (bzw. von den Werten) einer Variablen eine oder mehrere Handlungen auszuführen.

Anmerkung:

Falls unter einer bestimmten Bedingung keine Handlung ausgeführt werden soll, geben Sie das Statement IGNORE in der betreffenden Klausel des DECIDE ON-Statements an.

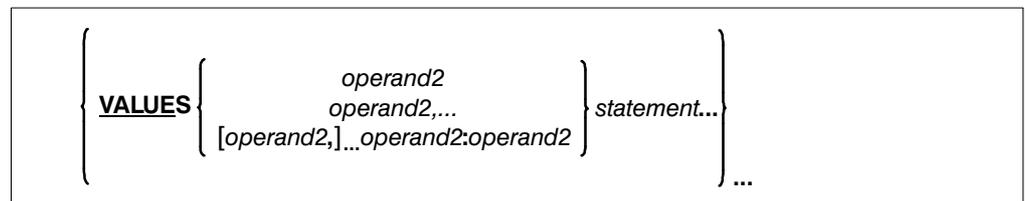
FIRST/EVERY

Mit einem dieser Schlüsselwörter geben Sie an, ob nur der *erste* gefundene Wert (FIRST) oder *alle* gefundenen Werte (EVERY) der Variablen verarbeitet werden sollen.

Kontrollfeld (*operand1*)

Als *operand1* geben Sie das Feld an, dessen Werte geprüft werden sollen.

VALUES-Klausel



Mit dieser Klausel geben Sie den Wert (*operand2*) des Kontrollfeldes an, sowie die *statements*, die ausgeführt werden sollen, wenn das Kontrollfeld diesen Wert hat.

Sie können einen Wert, mehrere Werte oder einen Bereich von Werten angeben, vor denen als Option einer oder mehrere Wert/e stehen können.

Werden mehrere Werte angegeben, müssen diese entweder mit dem Input-Delimiterzeichen (wie mit dem Session-Parameter ID definiert) oder mit einem Komma voneinander getrennt werden. Ein Komma darf hierzu allerdings nicht verwendet werden, falls das Komma als Dezimalkomma (mit dem Session-Parameter DC) definiert ist.

Bei einem Bereich von Werten geben Sie, durch einen Doppelpunkt voneinander getrennt, den Anfangs- und den Endwert des Bereiches an.

ANY

Mit ANY geben Sie die *statements* an, die ausgeführt werden sollen, wenn *irgendeiner* der in der VALUE-Klausel angegebenen Werte gefunden wird. Diese Statements werden *zusätzlich* zu den in der VALUE-Klausel angegebenen Statements ausgeführt.

ALL

Mit ALL geben Sie die *statements* an, die ausgeführt werden sollen, wenn *alle* in der VALUE-Klausel angegebenen Werte gefunden werden. Diese Statements werden *zusätzlich* zu den in der VALUE-Klausel angegebenen Statements ausgeführt.

Die ALL-Klausel kann nur in Verbindung mit dem Schlüsselwort EVERY eingesetzt werden.

NONE

Mit NONE geben Sie die *statements* an, die ausgeführt werden sollen, wenn *keiner* der angegebenen Werte gefunden wird.

Beispiel 1

```

/* EXAMPLE 'DECEX2': DECIDE ON (FIRST OPTION)
/*****
SET KEY ALL
INPUT 'TO UPDATE A RECORD, USE PF1 KEY' /
      'TO ADD    A RECORD, USE PF2 KEY' /
/*****
/* ROUTINE-UPD IS TO BE EXECUTED IF PF1 IS USED,
/* ROUTINE-ADD IS TO BE EXECUTED IF PF2 IS USED,
/* IF EITHER PF1 OR PF2 USED, END TRANSACTION IS TO BE EXECUTED,
/* IF NEITHER PF1 NOR PF2 ARE USED, NO STATEMENTS ARE TO BE EXECUTED.
/*****
DECIDE ON FIRST VALUE OF *PF-KEY
  VALUE 'PF1'
    PERFORM ROUTINE-UPD
  VALUE 'PF2'
    PERFORM ROUTINE-ADD
  ANY VALUE
    END TRANSACTION
    WRITE 'RECORD HAS BEEN MODIFIED'
  NONE VALUE
    IGNORE
END-DECIDE
/*****
END

```

Beispiel 2

```

/* EXAMPLE 'DECEX2E': DECIDE ON (EVERY OPTION)
/* THIS EXAMPLE SHOWS THE EFFECT OF USING THE EVERY CLAUSE
/*****
INPUT 'ENTER ANY VALUE BETWEEN 1 AND 9:' FIELD1(N1) (SG=OFF)
DECIDE ON EVERY VALUE OF FIELD1
  VALUE 1 : 4
    WRITE 'CONTENT OF FIELD1 IS 1-4'
  VALUE 2 : 5
    WRITE 'CONTENT OF FIELD1 IS 2-5'
  ANY VALUE
    WRITE 'CONTENT OF FIELD1 IS 1-5'
  ALL VALUE
    WRITE 'CONTENT OF FIELD1 IS 2-4'
  NONE VALUE
    WRITE 'CONTENT OF FIELD1 IS NOT 1-5'
END-DECIDE
/*****
END

```

```

ENTER ANY VALUE BETWEEN 1 AND 9: 4

```

```

Page      1

```

```

94-06-16 12:47:24

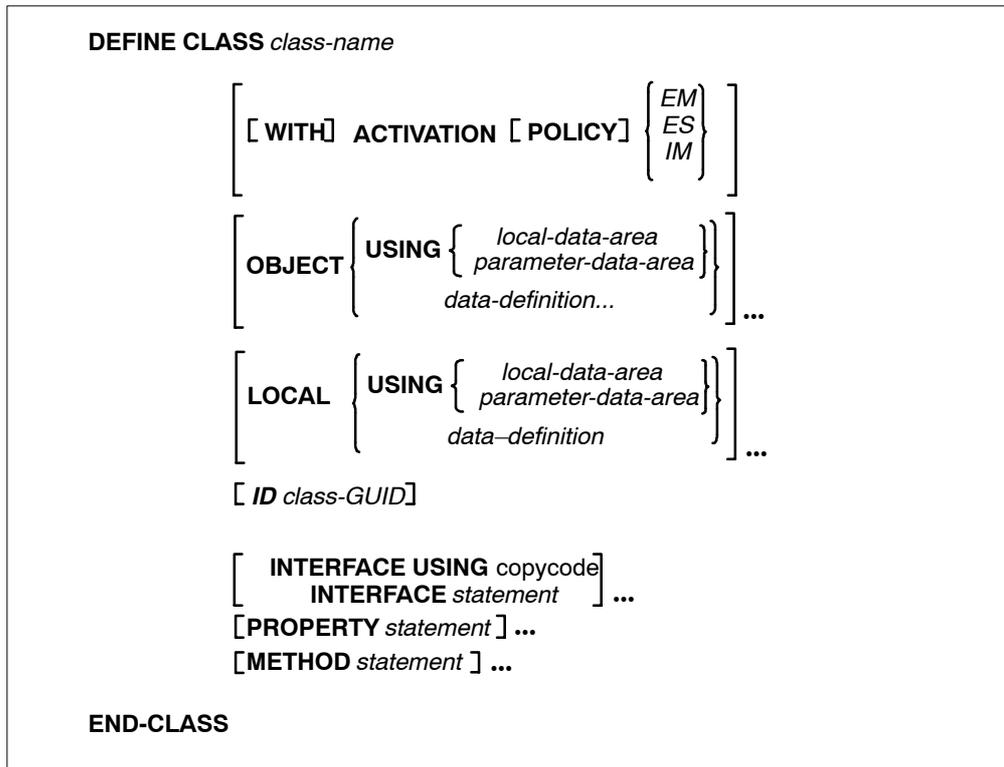
```

```

CONTENT OF FIELD1 IS 1-4
CONTENT OF FIELD1 IS 2-5
CONTENT OF FIELD1 IS 1-5
CONTENT OF FIELD1 IS 2-4

```

DEFINE CLASS



Funktion

Das Statement DEFINE CLASS dient dazu, eine Klasse innerhalb eines Natural Class-Moduls anzugeben.

Ein Natural Class-Modul besteht aus einem DEFINE CLASS-Statement gefolgt von einem END-Statement.

class-name

Dies ist der Name, der von Clients benutzt wird, um Objekte dieser Klasse zu erstellen. Er kann maximal bis zu 32 Zeichen lang sein und kann Punkte enthalten: dies führt dazu, daß es Klassen-Namen wie <company-name>.<application-name>.<class-name> geben kann. Jeder Bestandteil zwischen den Punkten (...) muß den Natural-Namenskonventionen für Benutzervariablen entsprechen (weitere Informationen siehe *Natural Referenzhandbuch*).

Wenn die Klasse von in verschiedenen Programmiersprachen geschriebenen Clients verwendet werden soll, sollte der Klassen-Name so gewählt werden, daß er nicht gegen die in diesen Sprachen geltenden Namenskonventionen verstößt. Bolero verwendet bspw. die Namenskonventionen von Java. Deshalb sollte eine in einem Bolero Client zu verwendende Klasse auch nicht gegen die Namenskonventionen von Java verstoßen.

WITH ACTIVATION POLICY-Klausel

Die WITH ACTIVATION POLICY-Klausel dient dazu, explizit die Aktivierungsmethode (“activation policy”) zu definieren, die für die aktuelle Klasse registriert ist.

Sie können die folgenden Parameter setzen:

Parameter	Beschreibung
EM	Setzt “Activation Policy” auf <i>ExternalMultiple</i> (extern-mehrfach).
ES	Setzt “Activation Policy” auf <i>ExternalSingle</i> (extern-einzeln).
IM	Setzt “Activation Policy” auf <i>Internal Multiple</i> (intern-mehrfach).

Wenn die Klasse in Source- und Objektform mit STOW gespeichert und registriert wird, überschreibt die Einstellung in der WITH ACTIVATION POLICY-Klausel den Profilparameter ACTPOLICY=*activation-policy* (für OS/390, DCOM=(ACTPOL=*activation-policy*)), wird aber wiederum bei der manuellen Registrierung mittels des REGISTER-Kommandos durch eine explizite “Activation Policy”-Definition überschrieben.

Weitere Informationen siehe den Unterabschnitt **Activation Policies** im Abschnitt **Distributing NaturalX Applications** in der *NaturalX Documentation*.

OBJECT-Klausel

Die OBJECT-Klausel dient dazu, Objektdaten zu definieren. Die Syntax der OBJECT-Klausel entspricht der für die LOCAL-Klausel des DEFINE DATA-Statements. Weitere Informationen siehe Beschreibung der LOCAL-Klausel des DEFINE DATA-Statements.

LOCAL-Klausel

Die LOCAL-Klausel dient dazu, weltweit eindeutige IDs (GUID = globally unique ID) in die Klassen-Definition aufzunehmen. GUIDs müssen nur definiert werden, wenn eine Klasse für DCOM registriert werden soll. GUIDs werden meistens in einer Local Data Area (LDA) definiert. Weitere Informationen siehe Abschnitt **Globally Unique Identifiers (GUIDs)**.

Die Syntax der LOCAL-Klausel entspricht der für die LOCAL-Klausel des DEFINE DATA-Statements. Weitere Informationen siehe Beschreibung der LOCAL-Klausel des DEFINE DATA-Statements.

ID-Klausel

Die ID-Klausel dient dazu, eine weltweit eindeutige ID (GUID = globally unique ID) für die Klasse zuzuweisen. Die Klasse GUID ist der Name einer GUID, die in der Data Area definiert worden ist, die in der LOCAL-Klausel enthalten ist. Die Klasse GUID ist eine (namentlich definierte) alphanumerische Konstante. Eine GUID muß einer Klasse zugewiesen werden, die für DCOM registriert werden soll.

INTERFACE USING-Klausel

Die INTERFACE USING-Klausel wird verwendet, um einen Copycode aufzunehmen, der INTERFACE-Statements enthält.

copycode

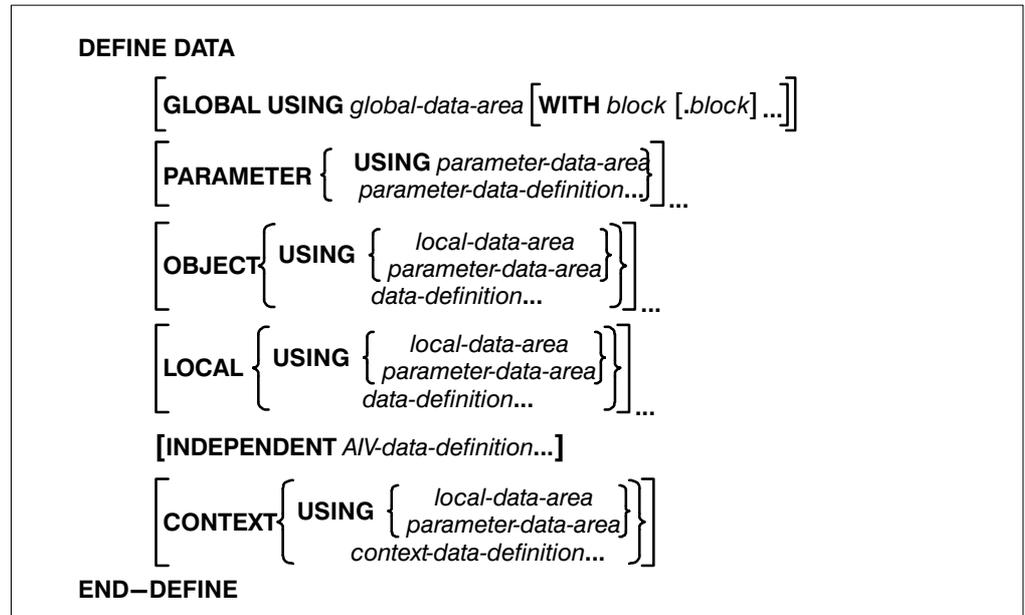
Der von der INTERFACE USING-Klausel verwendete Copycode kann eines oder mehrere INTERFACE-Statements enthalten.

Weitere Informationen siehe die folgenden Statements:

- INTERFACE
- PROPERTY
- METHOD.

DEFINE DATA

Syntax-Übersicht



Anmerkung:

Wird mehr als eine Klausel benutzt, so müssen die GLOBAL-, PARAMETER-, OBJECT-, LOCAL-, INDEPENDENT- und CONTEXT-Klauseln in der oben gezeigten Reihenfolge angegeben werden. Ein "leeres" DEFINE DATA-Statement ist nicht erlaubt, d.h. es muß mindestens eine Klausel (GLOBAL, PARAMETER, OBJECT, LOCAL, INDEPENDENT oder CONTEXT) angegeben werden, und es muß mindestens ein Feld definiert werden.

Funktion

Das Statement DEFINE DATA dient dazu, die Datenbereiche zu definieren, die von einem Natural-Programm benutzt werden. Wenn ein DEFINE DATA-Statement verwendet wird, muß es als erstes Statement in einem Programm/Unterprogramm stehen.

DEFINE DATA in Structured Mode

Im *Structured Mode* müssen alle verwendeten Variablen im DEFINE DATA-Statement definiert werden; sie dürfen nicht an anderer Stelle im Programm definiert werden.

DEFINE DATA in Reporting Mode

Im *Reporting Mode* ist das DEFINE DATA-Statement nicht zwingend erforderlich, da Variablen auch an anderer Stelle im Programm definiert werden können. Wenn Sie jedoch im *Reporting Mode* ein DEFINE DATA LOCAL-Statement verwenden, dürfen Sie an anderer Stelle im Programm keine weiteren Variablen (außer applikationsunabhängigen Variablen (AIVs)) definieren; wenn Sie im *Reporting Mode* ein DEFINE DATA INDEPENDENT-Statement verwenden, dürfen Sie an anderer Stelle im Programm keine weiteren AIVs definieren.

data-areas

Natural unterstützt drei Arten von Data Areas:

- global-data-area
- parameter-data-area
- local-data-area

global-data-area

Die in einer *global-data-area* definierten Datenelemente können von mehreren Programmierobjekten referenziert werden (wie im Kapitel **Objekttypen** des *Natural Leitfadens zur Programmierung* beschrieben). Die Global Data Area und die Objekte, die sie referenzieren, müssen sich in derselben Library (bzw. in einer Steplib) befinden.

Pro DEFINE DATA-Statement ist maximal eine *global-data-area* erlaubt.

parameter-data-area

In einer *parameter-data-area* werden Datenelemente definiert, die von einem Subprogramm, einer externen Subroutine oder einem Dialog als Parameter verwendet werden.

Den Parameter-Datenelementen dürfen weder Ausgangswerte noch konstante Werte zugeordnet werden; außerdem dürfen sie keine EM-, HD- oder PM-Definitionen besitzen. Es ist auch möglich, Parameter-Datenelemente im Subprogramm bzw. in der Subroutine selbst zu definieren. Innerhalb einer Helpoutine können auch Parameter definiert werden.

local-data-area

Die in einer *local-data-area* definierten Datenelemente können nur von einem einzigen Natural-Modul referenziert werden. (Statt in einer externen Local Data Area können die Datenelemente auch programm-/unterprogrammintern definiert werden.) Bei DEFINE DATA LOCAL kann auch der Name einer *parameter-data-area* angegeben werden.

Alle drei Arten von Data Areas können mit dem Data-Area-Editor erstellt und bearbeitet werden.

block

Datenblöcke (*blocks*) können einander bei der Ausführung eines Programms überlagern, wodurch sich der benötigte Speicherplatz verringert.

Die maximale Anzahl von Block-Levels ist 8 (einschließlich des Master-Blocks).

.block

.block-Notation(en) geben die in dem Programm verwendeten Blöcke an.

Nähere Informationen über Datenblöcke finden Sie im Abschnitt **Datenblöcke** des Kapitels **Felder definieren** im *Natural Leitfaden zur Programmierung*.

data-definition

$$\left\{ \begin{array}{l} \textit{level} \left\{ \begin{array}{l} \textit{group-name} [(\textit{array-definition})] \\ \textit{view-definition} \\ \textit{redefinition} \\ \textit{variable-definition} \\ \textit{handle-definition} \end{array} \right\} \end{array} \right\}$$

<i>level</i>	<p>Dies ist eine ein- oder zweistellige Zahl im Bereich von 01 bis 99 (die vorangestellte "0" ist nicht erforderlich), die in Verbindung mit der Gruppierung von Feldern verwendet wird. Felder mit einer Level-Nummer von 02 an aufwärts werden als Teil einer unmittelbar vorangehenden Gruppe mit einer jeweils nächstniedrigeren Level-Nummer betrachtet.</p> <p>Durch die Definition einer Gruppe (die auch nur aus einem Feld bestehen kann) ist es möglich, durch Angabe lediglich des Gruppennamens eine ganze Reihe von aufeinanderfolgenden Feldern gleichzeitig zu referenzieren.</p> <p>Eine Gruppe kann ihrerseits Teil einer anderen Gruppe sein. Bei der Vergabe von Level-Nummern für eine Gruppe darf kein Level ausgelassen werden.</p>
<i>group-name</i>	Der Name einer Gruppe. Der Name muß den für die Definition von Natural-Variablen geltenden Namenskonventionen entsprechen.

parameter-data-definition

$$\left\{ \begin{array}{l} \left\{ \begin{array}{l} \text{group-name [(array-definition)]} \\ \text{redefinition} \end{array} \right. \\ \text{level} \left\{ \begin{array}{l} \text{variable-name} \left\{ \begin{array}{l} \text{(format-length)} \\ \text{(format-length/array-definition)} \end{array} \right\} [\text{BY VALUE} [\text{RESULT}]] [\text{OPTIONAL}]] \\ \text{parameter-handle-definition} [\text{BY VALUE} [\text{RESULT}]] [\text{OPTIONAL}]] \end{array} \right. \end{array} \right\}$$

<i>level</i>	Ist das gleiche wie unter <i>data-definition</i> (siehe Seite 159).
<i>group-name</i>	Ist das gleiche wie unter <i>data-definition</i> (siehe Seite 159).
<i>variable-name</i>	Ist das gleiche wie unter <i>variable-definition</i> (siehe Seite 169).
<i>format-length</i>	Ist das gleiche wie unter <i>variable-definition</i> (siehe Seite 169).
DYNAMIC	Ein Parameter kann als DYNAMIC definiert werden. Weitere Informationen zur Verarbeitung dynamischer Variablen siehe den Abschnitt Large and Dynamic Variables/Fields im <i>Natural User's Guide for Windows NT</i> . Je nachdem, ob "Call-by-reference" oder "Call-by-value" verwendet wird, ist der entsprechende Übertragungsmechanismus gültig. Weitere Informationen siehe CALLNAT-Statement. <i>Anmerkung:</i> <i>DYNAMIC ist auf Großrechnern nicht verfügbar.</i>

BY VALUE	<p>Ohne BY VALUE (gilt standardmäßig) wird ein Parameter an ein(e) Subprogramm/Subroutine “by reference”, d.h. über seine Adresse, übergeben; das bedeutet, daß ein in einem CALLNAT- bzw. PERFORM-Statement als Parameter angegebenes Feld dasselbe Format und dieselbe Länge haben muß wie das entsprechende Feld in dem/der aufgerufenen Subprogramm / Subroutine.</p> <p>Mit BY VALUE wird ein Parameter “by value” an ein(e) Subprogramm/Subroutine übergeben; d.h. statt der Adresse des Parameters wird der <i>Wert</i> selbst übergeben. Das bedeutet, das Feld in dem/der Subprogramm/Subroutine braucht nicht dasselbe Format und dieselbe Länge haben wie der CALLNAT- / PERFORM- Parameter. Das Format und die Länge der beiden muß lediglich datenübertragungskompatibel entsprechend der in dem <i>Natural Referenzhandbuch</i> beschriebenen Regeln für Datenübertragung sein.</p> <p>Mit BY VALUE können Sie zum Beispiel die Länge eines Feldes in einem/einer Subprogramm/Subroutine vergrößern (falls eine Erweiterung des/der Subprogramms/Subroutine dies erforderlich machen sollte), ohne deswegen auch die Objekte, die das/die Subprogramm/Subroutine aufrufen, anpassen zu müssen.</p> <p>Bei Parameterdefinitionen für Dialoge (unter Windows und Windows NT) gilt folgendes:</p> <ul style="list-style-type: none"> – Ohne BY VALUE erfolgt die Übergabe eines in der Parameter Data Area innerhalb eines Dialogs angegebenen Parameters über seine Adresse; Format und Länge des Parameters, zum Beispiel in einem OPEN DIALOG- oder SEND-EVENT-Statement, müssen der Definition in der Parameter Data Area des Dialogs entsprechen. Sie können diese Art der Parameterübergabe in den Event-Handlern “Before Open” und “After Open” sowie in allen anderen Events verwenden, wenn die verwendeten Parameter in dem SEND EVENT-Statement, das den Event auslöst, übergeben werden. – BY VALUE bedeutet, daß der Parameterwert selbst übergeben wird; Format und Länge müssen nicht übereinstimmen; der Parameter im OPEN DIALOG- bzw. SEND EVENT-Statement muß nur mit dem Parameter im Dialog datenübertragungskompatibel sein.
-----------------	---

BY VALUE RESULT	<p>Während BY VALUE für die Übergabe eines Parameters <i>an</i> ein(e) Subprogramm/Subroutine gilt, bedeutet BY VALUE RESULT die Übergabe des Parameterwertes in beide Richtungen; d.h. der Parameterwert selbst wird vom aufrufenden Objekt an das/die Subprogramm/Subroutine übergeben, und bei der Rückkehr zum aufrufenden Objekt wird der Parameterwert selbst von dem/der Subprogramm/Subroutine an das aufrufende Objekt zurückgegeben.</p> <p>Bei BY VALUE RESULT müssen Format und Länge der betreffenden Felder in beide Richtungen datenübertragungskompatibel sein.</p> <p><i>Anmerkung:</i> <i>BY VALUE RESULT kann nicht in Dialogen verwendet werden.</i></p>
OPTIONAL	<p>Bei einem ohne OPTIONAL (Voreinstellung) definierten Parameter <i>muß</i> ein Wert vom aufrufenden Objekt übergeben werden. Bei einem mit OPTIONAL definierten Parameter kann ein Wert vom aufrufenden Objekt an diesen Parameter übergeben werden, es muß aber nicht unbedingt so sein. Im aufrufenden Objekt wird die Notation <i>nX</i> verwendet, um Parameter anzuzeigen, die übersprungen werden, d.h. für die keine Werte übergeben werden.</p> <p><i>Anmerkung:</i> <i>OPTIONAL ist auf Großrechnern nicht verfügbar.</i></p> <p>Mit der Option SPECIFIED können Sie zur Laufzeit herausfinden, ob ein optionaler Parameter definiert worden ist oder nicht.</p>

Beispiel für BY VALUE:

```

* Program
DEFINE DATA LOCAL
  1 #FIELDA (P5)
  ...
END-DEFINE
...
CALLNAT 'SUBR01' #FIELDA
...

* Subroutine SUBR01
DEFINE DATA PARAMETER
  1 #FIELDB (P9) BY VALUE
END-DEFINE
...

```

Beispiel für BY VALUE für Dialog:

```

/*Example of three parameters not passed BY VALUE:
1 #A (A10) /* Parameter Data Definition
1 #B (A20) /* of the called Dialog
1 #C (A30) /*
OPEN DIALOG 'MYDIALOG' #DLG$WINDOW WITH #X #Y #Z /* #X has to be A10,#Y has to
/* be A20,and #Z has to be A30

/*Example of three parameters passed BY VALUE:
1 #A (A10) BY VALUE /* Parameter Data Definition
1 #B (A20) BY VALUE /* of the called Dialog
1 #C (A30) BY VALUE /*
OPEN DIALOG 'MYDIALOG' #DLG$WINDOW WITH #X #Y #Z /* #X may be A1, #Y may be
/* A100,and #Z may be A253

```

parameter-handle-definition

<i>handle-name</i>	HANDLE OF { <i>dialog-element-type</i> OBJECT }
	(<i>array-definition</i>) HANDLE OF { <i>dialog-element-type</i> OBJECT }

handle-definition

<i>handle-name</i>	HANDLE OF { <i>dialog-element-type</i> OBJECT } [[CONSTANT INIT] <i>init-definition</i>]
	(<i>array-definition</i>) HANDLE OF { <i>dialog-element-type</i> OBJECT } [[CONSTANT INIT] <i>array-init-definition</i>]

Die Verwendung von “*handle-definition*” mit “*dialog-element-type*” ist nur unter Windows möglich.

<i>handle-name</i>	Der Name, den die Handle erhalten soll. Es gelten die Namenskonventionen für Benutzervariablen (siehe <i>Natural Referenzhandbuch</i>).
<i>dialog-element-type</i>	Der Typ des Dialog-Elements (nur möglich unter Windows). Mögliche Werte sind die Werte des Attributs TYPE. Näheres hierzu siehe Kapitel Dialog Elements bzw. Attributes der <i>Natural Dialog Components Documentation for Windows</i> .
OBJECT	Wird im Zusammenhang mit NaturalX benutzt (vgl. <i>NaturalX-Dokumentation</i>).

Die HANDLE-Definition im DEFINE DATA-Statement wird bei der Erstellung eines Dialogs oder Dialog-Elements automatisch generiert.

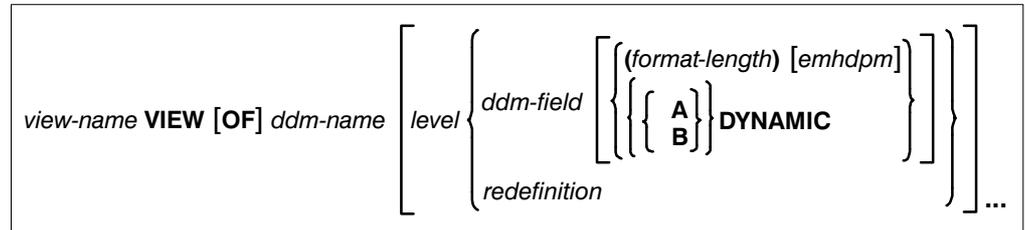
Nachdem Sie eine Handle definiert haben, können Sie den *handle-name* in jedem Statement verwenden, um die Attributwerte für den definierten *dialog-element-type* abzufragen, zu setzen oder zu ändern (siehe Kapitel **Event-Driven Programming Techniques** im *Natural User's Guide for Windows*).

Beispiele für *handle-definition*:

```
1 #SAVEAS-MENUITEM HANDLE OF MENUITEM  
1 #OK-BUTTON (1:10) HANDLE OF PUSHBUTTON
```

Anmerkung:

Wenn Sie “block”-Strukturen verwenden, dürfen Sie eine HANDLE OF OBJECT nur im Master-Block definieren, aber nicht in einem untergeordneten Block.

view-definition

Eine *view-definition* stellt einen Ausschnitt eines DDMs dar.

In einer Parameter Data Area ist eine *view-definition* nicht erlaubt.

<i>view-name</i>	Der Name, den der View erhalten soll. Es gelten die Namenskonventionen für Natural-Variablen.
<i>ddm-name</i>	Der Name des DDMs, von dem der View gebildet wird.
<i>level</i>	Dies ist eine ein- oder zweistellige Zahl im Bereich von 01 bis 99 (die vorangestellte "0" ist nicht erforderlich), die in Verbindung mit der Gruppierung von Feldern verwendet wird. Felder mit einer Level-Nummer von 02 an aufwärts werden als Teil einer unmittelbar vorangehenden Gruppe mit einer jeweils nächstniedrigeren Level-Nummer betrachtet. Durch die Definition einer Gruppe (die auch nur aus einem Feld bestehen kann) ist es möglich, durch Angabe lediglich des Gruppennamens eine ganze Reihe von aufeinanderfolgenden Feldern gleichzeitig zu referenzieren. Eine Gruppe kann ihrerseits Teil einer anderen Gruppe sein. Bei der Vergabe von Level-Nummern für eine Gruppe darf kein Level ausgelassen werden.
<i>ddm-field</i>	Der im verwendeten DDM definierte Name eines Feldes. Bei der Definition eines Views für ein HISTOGRAM-Statement darf dieser View lediglich den Deskriptor enthalten, den das HISTOGRAM-Statement benutzt.
<i>format-length</i>	Format und Länge des definierten Feldes. Werden diese Angaben nicht gemacht, wird die Format-/Längendefinition aus dem DDM übernommen. Im <i>Structured Mode</i> ist die Definition von Format und Länge nicht erlaubt. Sie können jedoch (zu Dokumentationszwecken) dieselbe Format-/Längenangabe hinschreiben, die im DDM definiert ist.
DYNAMIC	Definiert ein View-Feld als DYNAMIC. Weitere Informationen zum Verarbeiten von dynamischen Variablen siehe den Abschnitt Large and Dynamic Variables/Fields in Ihrer <i>Natural User's Guide Documentation</i> oder Ihrem <i>Natural Benutzerhandbuch</i> . <i>Anmerkung: DYNAMIC ist auf Großrechnern nicht verfügbar.</i>

redefinition

$$\text{REDEFINE } field\text{-name} \left\{ level \left\{ rfield \text{ (format-length) } \right. \right. \\ \left. \left. \text{FILLER } nX \right\} \right\} \dots$$

Eine *redefinition* dient dazu, eine Gruppe, einen View, ein einzelnes Feld bzw. eine einzelne Variable neu zu definieren.

<i>field-name</i>	Der Name der Gruppe bzw. des Feldes, das redefiniert werden soll.
<i>level</i>	Ist das gleiche wie unter <i>view-definition</i> .
<i>rfield</i>	Der Name des Feldes, das sich aus der Redefinition ergibt. <i>Anmerkung:</i> <i>Bei einer "redefinition" innerhalb einer "view-definition" darf für "rfield" kein Name vergeben werden, der schon als Feldname im zugrundeliegenden DDM existiert.</i>
<i>format-length</i>	Format und Länge von <i>rfield</i> .
FILLER nX	Mit dieser Notation können Sie in dem Feld, das redefiniert wird, <i>n</i> Füllbytes — d.h. Segmente, die nicht benutzt werden sollen — definieren. Die Definition von nachgestellten Füllbytes ist optional.

Anmerkungen:

Eine Handle kann nicht redefiniert werden. Eine Gruppe, die eine Handle enthält, kann nur bis unmittelbar vor der Handle — aber nicht darüber hinaus — redefiniert werden.

In einer "parameter-data-area" ist die "redefinition" einer Gruppe nur innerhalb eines REDEFINE-Blocks erlaubt; andernfalls könnte es bei der Übergabe von Parametern zwischen dem aufrufenden Programm und dem aufgerufenen Subprogramm zu internen Fehlern kommen.

REDEFINE — Beispiel 1:

```
DEFINE DATA LOCAL
  01 #VAR1 (A15)
  01 #VAR2
    02 #VAR2A (N4.1) INIT <0>
    02 #VAR2B (P6.2) INIT <0>
  01 REDEFINE #VAR2
    02 #VAR2RD (A10)
END-DEFINE
...
```

REDEFINE — Beispiel 2:

```
DEFINE DATA LOCAL
  01 MYVIEW VIEW OF STAFF
    02 NAME
    02 BIRTH
    02 REDEFINE BIRTH
      03 BIRTH-YEAR (N4)
      03 BIRTH-MONTH (N2)
      03 BIRTH-DAY (N2)
END-DEFINE
...
```

REDEFINE — Beispiel 3:

```
DEFINE DATA LOCAL
  1 #FIELD (A12)
  1 REDEFINE #FIELD
    2 #RFIELD1 (A2)
    2 FILLER 2X
    2 #RFIELD2 (A2)
    2 FILLER 4X
    2 #RFIELD3 (A2)
END-DEFINE
...
```

variable-definition

<i>variable-name</i>	}	$(format-length) \left[\begin{array}{c} \text{CONSTANT} \\ \text{INIT} \end{array} \right] \left[\begin{array}{c} \text{init-definition} \\ \text{array-init-definition} \end{array} \right] [emhdpm]$
		$(format-length/array-definition) \left[\begin{array}{c} \text{CONSTANT} \\ \text{INIT} \end{array} \right] [emhdpm]$
	}	$\left\{ \left\{ \begin{array}{c} A \\ B \end{array} \right\} \right\} \text{DYNAMIC}$

<i>variable-name</i>	Der Name, den die Variable erhalten soll. Es gelten die Regeln für Natural-Variablenamen. Informationen über Namenskonventionen für Benutzervariablen finden Sie im <i>Natural Referenzhandbuch</i> .
<i>format-length</i>	Format und Länge des definierten Feldes. Informationen über die Definition von Format und Länge von Benutzervariablen finden Sie im <i>Natural Referenzhandbuch</i> .
DYNAMIC	Ein Feld kann als DYNAMIC definiert werden. Weitere Informationen zur Verarbeitung von dynamischen Variablen finden Sie im Abschnitt Large and Dynamic Variables/Fields in Ihrer <i>Natural User's Guide Documentation</i> bzw. Ihrem <i>Natural Benutzerhandbuch</i> . <i>Anmerkung:</i> <i>DYNAMIC ist auf Großrechnern nicht verfügbar.</i>
CONSTANT	Die Variable bzw. das Array wird als namentlich definierte Konstante behandelt. Der zugeordnete konstante Wert bzw. die Werte werden jedesmal verwendet, wenn die Variable bzw. das Array referenziert wird. Die Werte können während der Ausführung des Programms nicht geändert werden. <i>Anmerkung:</i> <i>Aus Gründen der internen Verarbeitung dürfen innerhalb einer Gruppendifinition nicht gleichzeitig Variablen und Konstanten definiert werden, d.h. eine Gruppe darf entweder nur Variablen oder nur Konstanten enthalten.</i>
INIT	Der Variablen bzw. dem Array wird ein Ausgangswert (initial value) zugewiesen. Dieser Wert wird auch benutzt, wenn die Variable bzw. das Array in einem RESET INITIAL-Statement referenziert wird.

Die folgenden Beschränkungen gelten für eine dynamische Variable:

- Die Klauseln INIT und CONST sind nicht zulässig.
- Eine dynamische Variable kann nicht als Array definiert werden.
- Eine Redefinition einer dynamischen Variable ist nicht zulässig.
- Eine dynamische Variable kann nicht in einer Redefinition enthalten sein.

Standard-Ausgangswerte

Wenn Sie keine INIT- oder CONSTANT-Angaben machen, wird ein Feld je nach Format mit einem Standard-Ausgangswert initialisiert:

Format	Standard-Ausgangswert
B, F, I, N, P	0
A	Leerzeichen
L	F(ALSE)
D	D' '
T	T'00:00:00'
C	(AD=D)
GUI-Handle	NULL-HANDLE
Objekt-Handle	NULL-HANDLE

init-definition

$\left\{ \begin{array}{l} \text{<constant>} \\ \text{<system-variable>} \\ \text{FULL LENGTH <character-s>} \\ \text{LENGTH } n \text{ <character-s>} \end{array} \right\}$

<i>constant</i>	Der konstante Wert, mit der die Variable initialisiert werden soll, bzw. der konstante Wert, der dem Feld fest zugewiesen wird. Nähere Informationen zu Konstanten siehe <i>Natural Referenzhandbuch</i> .
<i>system-variable</i>	Als Ausgangswert einer Variablen können Sie auch den Wert einer Natural-Systemvariablen benutzen. <i>Anmerkung:</i> Wenn die Variable in einem RESET INITIAL-Statement referenziert wird, wird die Systemvariable neu ausgewertet; d.h. die Variable wird nicht auf den Wert zurückgesetzt, den die Systemvariable zu Beginn der Programmausführung hatte, sondern auf den Wert, den sie zum Zeitpunkt der Ausführung des RESET INITIAL-Statements hat.
FULL LENGTH LENGTH <i>n</i>	Als Ausgangswert können Sie eine Variable auch, vollständig oder teilweise, mit einem bestimmten Zeichen oder einer Zeichenkette füllen (nur bei alphanumerischen Variablen möglich). Mit der Option "FULL LENGTH" füllen Sie das gesamte Feld mit dem/den angegebenen Zeichen (<i>character(s)</i>). Mit der Option "LENGTH <i>n</i> " füllen Sie die ersten <i>n</i> Stellen des Feldes mit dem/den angegebenen Zeichen (<i>character(s)</i>). <i>n</i> muß eine numerische Konstante sein.

Beispiel für Systemvariable als Ausgangswert:

```
DEFINE DATA LOCAL
1 #MYDATE (D) INIT <*DATX>
END-DEFINE
```

Beispiel für FULL LENGTH:

In diesem Beispiel wird das gesamte Feld mit Sternen gefüllt.

```
DEFINE DATA LOCAL
1 #FIELD (A25) INIT FULL LENGTH <'*'>
END-DEFINE
```

Beispiel für LENGTH n:

In diesem Beispiel werden die ersten 4 Stellen des Feldes mit Ausrufezeichen gefüllt.

```
DEFINE DATA LOCAL
1 #FIELD (A25) INIT LENGTH 4 <'!'>
END-DEFINE
```

array-definition

- Variable_Arrays_in_einer_Parameter_Data_Area

$\{index[:index]\}, \dots_3$

Mit einer *array-definition* definieren Sie die Dimensionen (Indexe) eines Arrays.

<i>index</i>	Dies kann eine ganzzahlige numerische Konstante, eine vorher namentlich definierte Konstante oder (für Datenbank-Arrays) eine vorher definierte Benutzervariable sein.
--------------	--

Zur Definition von Arrays vgl. Abschnitt **Index-Notation** im *Natural Referenzhandbuch*.

Beispiele für Array-Definitionen:

```
#ARRAY1(A5/3)           /* a one-dimensional array
#ARRAY2(A5/1:5,1:5)     /* a two-dimensional array
#ARRAY3(A5/1:10,1:10,1:10) /* a three-dimensional array
```

Variable Arrays in einer Parameter Data Area

In einer Parameter Data Area können Sie ein Array mit einer variablen Anzahl von Ausprägungen angeben, und zwar mit der Index-Notation “1:V”. Zum Beispiel:

```
#ARRAYX (A5/1:V)
```

```
#ARRAYY (I2/1:V,1:V)
```

Ein Array, das mit dem Index “1:V” definiert wird, darf weder redefiniert werden noch das Ergebnis einer Redefinition sein. Da die Anzahl der Ausprägungen bei der Kompilierung nicht bekannt ist, darf das Array nicht mit der Index-Notation (*) referenziert werden, außer in einem der folgenden Statements: ADD, COMPRESS, COMPUTE, DISPLAY, DIVIDE, EXAMINE, IF, MULTIPLY, RESET, SUBTRACT.

Ein variables Array darf nur insgesamt (d.h. alle seine Ausprägungen) oder als Skalarwert (d.h. eine einzige Ausprägung) referenziert werden. Zum Beispiel:

```
#ARRAYX (*)
#ARRAYY (*,*)
#ARRAYX (1)
#ARRAYY (5,#FIELDX)
```

Ein Teilbereich eines variablen Arrays darf nicht referenziert werden:

```
#ARRAYY (1,*) /* nicht erlaubt
```

Um Laufzeitfehler zu vermeiden, sollten Sie die maximale Anzahl der Ausprägungen eines solchen Arrays in einem anderen Parameter an das Subprogramm bzw. die Subroutine übergeben.

Anmerkungen:

Wenn Sie eine Parameter Data Area, die einen Index “1:V” enthält, als Local Data Area verwenden (d.h. in einem DEFINE DATA LOCAL-Statement angeben), müssen Sie vorher eine Variable namens “V” als CONSTANT definiert haben.

In einem Dialog kann ein Index “1:V” nicht in Verbindung mit BY VALUE verwendet werden.

Siehe auch Systemvariable *OCCURRENCE im *Natural Referenzhandbuch*.

array-init-definition

$$\left\{ \left[\left(\begin{array}{c} \text{ALL} \\ \{ \text{index} [\text{:index}] \} \\ \text{V} \end{array} \right) , \dots, 3 \right] \left\{ \left\{ \begin{array}{l} \text{FULL LENGTH} \\ \text{LENGTH } n \end{array} \right\} \langle \text{character-s}, \dots \rangle \right\} \left\{ \langle \text{constant} \rangle \right\} \left\{ \langle \text{system-variable} \rangle , \dots \right\} \right\} \dots$$

Mit dieser Klausel definieren Sie die Ausgangswerte bzw. konstanten Werte, die einem Array zugewiesen werden.

Für ein redefiniertes Feld ist eine *array-init-definition* nicht erlaubt.

ALL	Alle Ausprägungen in allen Dimensionen des Arrays werden mit dem <i>gleichen</i> Wert initialisiert.
<i>index</i>	Nur die im <i>index</i> angegebenen Ausprägungen des Arrays werden initialisiert. Wenn Sie einen <i>index</i> angeben, dürfen Sie mit <i>constant</i> nur einen einzigen Wert angeben; d.h. die angegebenen Ausprägungen werden mit dem gleichen Wert initialisiert.
V	Diese Notation ist nur relevant bei mehrdimensionalen Arrays, wenn den Ausprägungen einer Dimension <i>unterschiedliche</i> Werte zugewiesen werden. “V” bezeichnet einen Indexbereich, der sich über alle Ausprägungen der mit “V” bezeichneten Dimension erstreckt, d.h. alle Ausprägungen in dieser Dimension werden initialisiert. Pro Array darf höchstens eine Dimension mit “V” bezeichnet werden. Die Ausprägungen werden Ausprägung für Ausprägung mit den für diese Dimension angegebenen Werten initialisiert. Die Anzahl der Werte darf nicht größer sein als die Anzahl der Ausprägungen der mit “V” bezeichneten Dimension.
<i>constant</i>	Der konstante Wert, der dem Array entweder als Ausgangswert (INIT) oder als konstanter Wert (CONSTANT) zugewiesen wird. Weitere Informationen zur Definition von Konstanten finden Sie im <i>Natural Referenzhandbuch</i> . <i>Anmerkung:</i> <i>Ausprägungen, für die Sie keine Werte angeben, werden mit einem Standardwert (siehe Seite 170) initialisiert.</i>

<i>system-variable</i>	<p>Als Ausgangswert können Sie einem Array auch den Wert einer Natural-Systemvariablen zuweisen.</p> <p><i>Anmerkung:</i> <i>Wenn Sie mehrere constants/system-variables angeben, müssen Sie diese entweder mit dem Input-Delimiterzeichen (wie mit dem Session-Parameter ID definiert) oder mit einem Komma voneinander trennen. Ein Komma darf hierzu allerdings nicht verwendet werden, falls das Komma als Dezimalkomma (mit dem Session-Parameter DC) definiert ist.</i></p>
FULL LENGTH LENGTH <i>n</i>	<p>Als Ausgangswert können Sie ein Array auch, vollständig oder teilweise, mit einem bestimmten Zeichen oder einer Zeichenkette füllen (nur bei alphanumerischen Arrays möglich):</p> <p>Mit “FULL LENGTH” füllen Sie die Array-Ausprägungen vollständig mit dem/den angegebenen Zeichen (<i>character(s)</i>).</p> <p>Mit “LENGTH <i>n</i>” füllen Sie die ersten <i>n</i> Stellen der Array-Ausprägungen mit dem/den angegebenen Zeichen (<i>character(s)</i>).</p> <p>Eine <i>system-variable</i> darf mit “FULL LENGTH” oder “LENGTH <i>n</i>” nicht angegeben werden.</p> <p>Innerhalb einer <i>array-init-definition</i> können Sie nur entweder “FULL LENGTH” oder “LENGTH <i>n</i>” angeben, aber nicht beides.</p>

Beispiel für LENGTH *n* für ein Array:

Hier werden die ersten 5 Stellen jeder Ausprägung des Arrays mit “NONON” gefüllt.

```

DEFINE DATA LOCAL
1 #FIELD (A25/1:3) INIT ALL LENGTH 5 <'NO'>
...
END-DEFINE

```

Zahlreiche Beispiele für die Zuweisung von Ausgangswerten für Arrays finden Sie im *Natural Leitfaden zur Programmierung*.

emhdpm

```
([EM=value] [HD='value'] [PM=value])
```

Mit dieser Option können Sie für ein/e Feld/Variable zusätzliche Parameter definieren.

EM=value	Mit diesem Parameter können Sie eine Editiermaske definieren. Siehe Session-Parameter EM im <i>Natural Referenzhandbuch</i> .
HD='value'	Mit diesem Parameter können Sie eine Überschrift definieren, die als Standard-Spaltenüberschrift für das Feld ausgegeben wird (siehe DISPLAY-Statement).
PM=value	Mit diesem Parameter können Sie den Print-Modus setzen, der definiert, wie Felder ausgegeben werden. Siehe Session-Parameter PM im <i>Natural Referenzhandbuch</i> .

Wenn Sie für ein Datenbankfeld weder eine Editiermaske (EM=) noch eine Spaltenüberschrift (HD=) angeben, werden die Standard-Editiermaske und die Standard-Spaltenüberschrift aus dem DDM genommen. Wenn sie jedoch eins von beiden angeben, wird für das jeweils andere *nicht* die Standarddefinition aus dem DDM genommen.

AIV-data-definition

```
level { AIV-definition  
REDEFINE field-name }
```

Mit DEFINE DATA INDEPENDENT können Sie applikationsunabhängige Variablen (application-independent variables, AIVs) definieren.

<i>level</i>	Dies ist eine ein- oder zweistellige Zahl im Bereich von 01 bis 99 (die vorangestellte "0" ist nicht erforderlich), die in Verbindung mit der Gruppierung von Feldern verwendet wird. Eine AIV muß auf Level 01 definiert werden. Andere Levels werden nur in einer Redefinition verwendet.
REDEFINE	Mit REDEFINE können Sie eine AIV redefinieren. Die aus der Redefinition resultierenden Felder dürfen keine AIVs sein.

AIV-definition

$$\text{variable-name} \left\{ \begin{array}{l} (\text{format-length}) \text{ [INIT } \textit{init-definition}] \\ (\text{format-length/array-definition}) \text{ [INIT } \textit{array-init-definition}] \end{array} \right\} [\textit{emhdpm}]$$

<i>variable-name</i>	Der Name, den die applikationsunabhängige Variable erhalten soll. Das erste Zeichen des Namens muß ein “+” sein. Es gelten die Regeln für Natural-Variablenamen. Informationen über Namenskonventionen für Benutzervariablen finden Sie im <i>Natural Referenzhandbuch</i> .
<i>format-length</i>	Format und Länge des Feldes. Informationen über die Definition von Format und Länge von Benutzervariablen finden Sie im <i>Natural Referenzhandbuch</i> .
INIT	Der applikationsunabhängigen Variablen wird ein Ausgangswert (initial value) zugewiesen. Dieser Wert wird auch benutzt, wenn die Variable in einem RESET INITIAL-Statement referenziert wird.
<i>init-definition</i>	Ist das gleiche wie unter <i>variable-definition</i> (siehe Seite 169).
<i>array-definition</i>	Ist das gleiche wie unter <i>variable-definition</i> (siehe Seite 169).
<i>array-init-definition</i>	Ist das gleiche wie unter <i>variable-definition</i> (siehe Seite 169).
<i>emhdpm</i>	Ist das gleiche wie unter <i>variable-definition</i> (siehe Seite 169).

context-data-definition

<i>level</i>	$\left\{ \begin{array}{l} \textit{variable-definition} \\ \textit{redefinition} \\ \textit{handle-definition} \end{array} \right\}$
--------------	---

DEFINE DATA CONTEXT wird im Zusammenhang mit Natural Remote Procedure Call (RPC) verwendet. Es dient dazu, sogenannte “Kontextvariablen” zu definieren, d.h. Variablen, die mehreren Remote-Subprogrammen innerhalb einer Conversation zur Verfügung stehen, ohne daß diese Variablen explizit als Parameter mit den betreffenden CALLNAT-Statements übergeben werden müssen. Es genügt, die Kontextvariablen in jedem Subprogramm, in dem sie zur Verfügung stehen sollen, in einem DEFINE DATA CONTEXT-Statement zu definieren.

Eine Kontextvariable wird namentlich referenziert, und ihr Inhalt steht allen Subprogrammen zur Verfügung, die den Variablennamen innerhalb einer Conversation referenzieren.

Nähere Informationen siehe Beschreibung von Natural RPC in Ihrer *Natural RPC Documentation*.

<i>level</i>	Siehe Seite 159.
<i>variable-definition</i>	Siehe Seite 169.
<i>redefinition</i>	Siehe Seite 167.
handle-definition	Siehe Seite 164.

DEFINE DATA OBJECT

DEFINE DATA OBJECT wird im Zusammenhang mit NaturalX verwendet und ist in der *NaturalX-Dokumentation* beschrieben.

Qualifizieren von Datenstrukturen

Um beim Referenzieren eines Feldes das Feld zu identifizieren, können Sie es qualifizieren; d.h. Sie geben vor dem Feldnamen den Namen des Level-1-Datenelements an, in dem das Feld enthalten ist, gefolgt von einem Punkt.

Wenn ein Feld durch seinen Namen allein nicht eindeutig identifiziert werden kann (zum Beispiel, wenn derselbe Feldname in mehreren Gruppen/Views verwendet wird), müssen Sie das Feld beim Referenzieren qualifizieren.

Die Kombination aus Level-1-Datenelement und Feldname muß eindeutig sein (siehe erstes Beispiel unten).

Der Qualifizierer muß ein auf Level 1 definiertes Datenelement sein (siehe zweites Beispiel unten).

Beispiel:

```
DEFINE DATA LOCAL
1 FULL-NAME
  2 LAST-NAME (A20)
  2 FIRST-NAME (A15)
1 OUTPUT-NAME
  2 LAST-NAME (A20)
  2 FIRST-NAME (A15)
END-DEFINE
...
MOVE FULL-NAME.LAST-NAME TO OUTPUT-NAME.LAST-NAME
...
```

Beispiel:

```
DEFINE DATA LOCAL
1 GROUP1
  2 SUB-GROUP
    3 FIELD1 (A15)
    3 FIELD2 (A15)
END-DEFINE
...
MOVE 'ABC' TO GROUP1.FIELD1
...
```

Anmerkung:

Falls Sie für eine Benutzervariable und ein Datenbankfeld denselben Namen verwenden (was Sie ohnehin nicht tun sollten), müssen Sie das Datenbankfeld qualifizieren, wenn Sie es referenzieren wollen; sonst wird stattdessen die Benutzervariable referenziert.

Beispiel 1

```
/* EXAMPLE 'DDAEX1': DEFINE DATA
/*****
DEFINE DATA LOCAL
01 #VAR1 (A15)
01 #VAR2
    02 #VAR2A (N4.1) INIT <1111>
    02 #VAR2B (N6.2) INIT <22222>
01 REDEFINE #VAR2
    02 #VAR2C (A2)
    02 #VAR2D (A2)
    02 #VAR2E (A6)
END-DEFINE
/*****
WRITE NOTITLE '=' #VAR2A / '=' #VAR2B /
              '=' #VAR2C / '=' #VAR2D / '=' #VAR2E
/*****
END
```

```
#VAR2A: 1111.0
#VAR2B: 222222.00
#VAR2C: 11
#VAR2D: 11
#VAR2E: 022222
```

Beispiel 2

```

/* EXAMPLE 'DDAEX2': DEFINE DATA (ARRAY DEFINITION/INITIALIZATION)
/*****
DEFINE DATA LOCAL
01 #VAR1 (A1/1:2,1:2) INIT (1,V) <'A','B'>
01 #VAR2 (N5/1:2,1:3) INIT (1,2) <200>
01 #VAR3 (A1/1:4,1:3) INIT (V,2:3) <'W','X','Y','Z'>
END-DEFINE
/*****
WRITE NOTITLE '=' #VAR1 (1,1) '=' #VAR1 (1,2)
           / '=' #VAR1 (2,1) '=' #VAR1 (2,2)
/*****
WRITE ///   '=' #VAR2 (1,1) '=' #VAR2 (1,2)
           / '=' #VAR2 (2,1) '=' #VAR2 (2,2)
/*****
WRITE ///   '=' #VAR3 (1,1) '=' #VAR3 (1,2) '=' #VAR3 (1,3)
WRITE      / '=' #VAR3 (2,1) '=' #VAR3 (2,2) '=' #VAR3 (2,3)
WRITE      / '=' #VAR3 (3,1) '=' #VAR3 (3,2) '=' #VAR3 (3,3)
WRITE      / '=' #VAR3 (4,1) '=' #VAR3 (4,2) '=' #VAR3 (4,3)
/*****
END

```

```

#VAR1: A #VAR1: B
#VAR1:   #VAR1:

#VAR2:      0 #VAR2:      200
#VAR2:      0 #VAR2:      0

#VAR3:   #VAR3: W #VAR3: W
#VAR3:   #VAR3: X #VAR3: X
#VAR3:   #VAR3: Y #VAR3: Y
#VAR3:   #VAR3: Z #VAR3: Z

```

Beispiel 3

```

/* EXAMPLE 'DDAEX3': DEFINE DATA (VIEW DEFINITION, REDEFINE ARRAY)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE (A20/2)
  2 PHONE
1 #ARRAY (A75/1:4)
1 REDEFINE #ARRAY
  2 #ALINE (A25/1:4,1:3)
1 #X (N2) INIT <1>
1 #Y (N2) INIT <1>
END-DEFINE
/*****
FORMAT PS=20
LIMIT 5
FIND EMPLOY-VIEW WITH NAME = 'SMITH'
  MOVE NAME TO #ALINE (#X,#Y)
  MOVE ADDRESS-LINE(1) TO #ALINE (#X+1,#Y)
  MOVE ADDRESS-LINE(2) TO #ALINE (#X+2,#Y)
  MOVE PHONE TO #ALINE (#X+3,#Y)
  IF #Y = 3
    RESET INITIAL #Y
    PERFORM PRINT
  ELSE
    ADD 1 TO #Y
  END-IF
  AT END OF DATA
    PERFORM PRINT
  END-ENDDATA
END-FIND
/*****
DEFINE SUBROUTINE PRINT
WRITE NOTITLE (AD=OI) #ARRAY(*)
RESET #ARRAY(*)
SKIP 1
END-SUBROUTINE
/*****
END

```

SMITH ENGLANDSVEJ 222 554349	SMITH 3152 SHETLAND ROAD MILWAUKEE (414)877-4563	SMITH 14100 ESWORTHY RD. MONTERREY (408)994-2260
SMITH 5 HAWTHORN OAK BROOK (312)150-9351	SMITH 2307 DARIUS LANE TAMPA (813)131-4010	

Beispiel 4

```

/* EXAMPLE 'DDAEX4': DEFINE DATA (GLOBAL, PARAMETER AND LOCAL AREAS)
/*****
/* MAIN PROGRAM
/*****
DEFINE DATA GLOBAL USING GLOBAL-1
      LOCAL
        1 #FIELD1 (A10)
        1 #FIELD2 (N5)
END-DEFINE
/*****
/* ...
CALLNAT 'SUBP1' #FIELD1 #FIELD2
/* ...
END

/* SUBPROGRAM 'SUBP1'
DEFINE DATA PARAMETER
  1 #FIELDA (A10)
  1 #FIELDB (N5)
END-DEFINE
/*****
/* ...
END

```

Beispiel 5

```

* EXAMPLE 'DDAEX5': DEFINE DATA (INITIALIZATION)
*****
DEFINE DATA LOCAL
  1 #START-DATE (D)   INIT <*DATE>
  1 #UNDERLINE   (A50) INIT FULL LENGTH <'_'>
  1 #SCALE       (A65) INIT LENGTH 65 <'...+.../'>
END-DEFINE
*
WRITE NOTITLE #START-DATE (DF=L)
      / #UNDERLINE
      / #SCALE
END

```

<pre> 1999-01-19 ----- ...+.../...+.../...+.../...+.../...+.../...+.../...+ </pre>
--

Beispiel 6

```

/* EXAMPLE 'DDAEX6': DEFINE DATA (VARIABLE ARRAY)
/*****
DEFINE DATA
  PARAMETER
    01 #STRING (A1/1:V)
    01 #MAX (P3)
  LOCAL
    01 #I (P3)
END-DEFINE
/*****
FOR #I = 1 TO #MAX
  IF #STRING (#I) < H'40'
    MOVE H'40' TO #STRING (#I)
  END-IF
END-FOR
END

```

Beispiel 7

```
DEFINE DATA LOCAL
1 #MyHomePage (A4096) /* alpha variable with max. 4096 characters
1 #MyStream (B1000000/1:10) /* binary array with 10 occurrences and max.
/* 1000000 bytes per occ.
1 #MyDynHomePage (A) DYNAMIC /* dynamic alpha variable
1 #MyDynStream (B) DYNAMIC /* dynamic binary variable
END-DEFINE
```

DEFINE PRINTER

```
DEFINE PRINTER ([logical-printer-name=]n)
```

```
[OUTPUT operand1]
```

```
[
  PROFILE operand2
  FORMS operand2
  NAME operand2
  DISP operand2
  CLASS operand2
  COPIES operand3
  PRTY operand4
]...7
```

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	ja	nein
Operand2	C S	A	ja	nein
Operand3	C S	N	ja	nein
Operand4	C S	N P I	ja	nein

Verwandte Statements

CLOSE PRINTER, DISPLAY, WRITE.

Funktion

Das Statement DEFINE PRINTER dient dazu, einer Report-Nummer einen symbolischen Namen zuzuordnen und die Zuweisung eines Reports zu einem logischen Bestimmungsort (Drucker) zu steuern. Dies bietet zusätzliche Flexibilität bei der Erstellung von Ausgaben für verschiedene logische Drucker-Warteschlangen.

Ist bei der Ausführung dieses Statements der angegebene Drucker bereits offen, bewirkt dieses Statement implizit, daß der Drucker geschlossen wird. Um einen Drucker explizit zu schließen, sollten Sie das Statement CLOSE PRINTER verwenden.

Drucker

Als *logical-printer-name* geben Sie den logischen Namen an, den Drucker *n* erhalten soll. Der vergebene Name wird auch bei der “*rep*”-Notation in einem DISPLAY- bzw. WRITE-Statement verwendet. Für logische Namen gelten die gleichen Namenskonventionen wie für Benutzervariablen (siehe *Natural Referenzhandbuch*).

Die Druckernummer *n* darf einen Wert von 1 bis 31 haben. Beim Einsatz von Natural Advanced Facilities darf *n* ebenfalls einen Wert von 1 bis 31 haben.

Einer Druckernummer dürfen mehrere verschiedene logische Namen gegeben werden.

Im Gegensatz zum Ausgabeziel (s. unten) wird *logical-printer-name=n* zur *Kompilierungszeit* ausgewertet und ist daher unabhängig vom Programmkontrollablauf.

OUTPUT *operand1*

- Drucker unter OS/390 mit Zugriffsmethode AM=STD (Standard-Batch)
- Drucker unter VM/CMS mit Zugriffsmethode AM=STD (Standard-Batch)
- Drucker unter BS2000/OSD mit Zugriffsmethode AM=STD (Standard-Batch)
- Drucker unter Com-plete
- Drucker unter Natural Advanced Facilities
- Zusätzliche Reports.

Mit *operand1* geben Sie das Ausgabeziel innerhalb des Online-Spooling-Systems an. Diese Angabe wird zur *Laufzeit* ausgewertet.

Ist *operand1* eine Variable, muß sie mit Format/Länge A8 definiert sein.

Es kann ein beliebiger logischer Druckername angegeben werden, vorausgesetzt dieser ist in der/dem verwendeten Parameterdatei/Parametermodul bzw. beim Starten von Natural über Parameter oder JCL definiert worden.

Unter OpenVMS, UNIX und Windows muß der Name als LPT*nn* angegeben werden, wobei *nn* 1 bis 31 sein darf.

Beispiele:

LPT12	(OpenVMS-, UNIX- bzw. Windows-Druckername)
CMPRT01	(OS/JCL DDNAME)
P01	(BS2000/OSD-Dateikommando)
TID111	(Com-plete-Druckername)
PCPRNT01	(für Natural Connection definierter Drucker)

Wenn Sie die OUTPUT-Klausel weglassen, wird das Ausgabeziel durch das “Device Assignment” in der verwendeten Parameterdatei — bzw. auf Großrechnern durch den Profilparameter PRINT bzw. das Makro NTPRINT — bestimmt (siehe *Natural Reference Documentation*).

Drucker unter OS/390 mit Zugriffsmethode AM=STD (Standard-Batch)

Unter OS/390 können Sie für einen Drucker, der (entweder automatisch in der JCL oder im NTPRINT-Makro des Natural-Parametermoduls bzw. mit dem dynamischen Profilparameter PRINT) mit Zugriffsmethode AM=STD definiert ist, mit *operand1* einen logischen oder einen physischen Dataset-Namen angeben, der dieser Druckernummer zugewiesen werden soll.

In diesem Fall kann *operand1* 1 bis 253 Stellen lang sein und eins der folgenden sein:

- ein logischer Dataset-Name (DD-Name, 1 bis 8 Stellen)
- ein physischer Dataset-Name eines katalogisierten Datasets (1 bis 44 Stellen) oder ein physischer Dataset-Member-Name (1 bis 44 Stellen für den Dataset-Namen plus 1 bis 8 Stellen in Klammern für den Member-Namen)
- ein Pfad- und Member-Name einer HFS-Datei (1 bis 253 Stellen) in einer MVS-UNIX-Services-Umgebung
- eine JES-Spoolfile-Klasse
- "NULLFILE" (bezeichnet ein Dummy-Dataset).

Logische Dataset-Namen

Zum Beispiel:

```
DEFINE PRINTER (21) OUTPUT 'SYSPRINT'
```

Das angegebene Dataset muß zugewiesen worden sein, bevor das DEFINE PRINTER-Statement ausgeführt wird.

Die Zuweisung kann über JCL, CLIST oder dynamische Zuweisung (SVC 99) erfolgen. Für dynamische Zuweisung können Sie den User Exit USR2021 in Library SYSEXT verwenden.

Der im DEFINE PRINTER-Statement angegebene Dataset-Name überschreibt den im Subparameter DEST des NTPRINT-Makros bzw. PRINT-Profilparameters angegebenen Namen.

Optional kann dem Dataset-Namen "DDN=" vorangestellt werden, um anzuzeigen, daß es sich um einen DD-Namen handelt, und Namenskonflikte mit zusätzlichen Reports zu vermeiden. Zum Beispiel:

```
DEFINE PRINTER (22) OUTPUT 'DDN=SOURCE'
```

Physische Dataset-Namen

Zum Beispiel:

```
DEFINE PRINTER (23) OUTPUT 'TEST.PRINT.FILE'
```

Das angegebene Dataset muß in katalogisierter Form vorhanden sein. Wenn das DEFINE PRINTER-Statement ausgeführt wird, wird das Dataset dynamisch über SVC 99 mit dem aktuellen DD-Namen und der Option DISP=SHR zugewiesen.

Wenn der Dataset-Name 8 Stellen oder kürzer ist und keinen Punkt “.” enthält, könnte er als DD-Name mißinterpretiert werden. Um dies zu vermeiden, stellen Sie ihm “DSN=” voran. Zum Beispiel:

```
DEFINE PRINTER (22) OUTPUT 'DSN=PRINTXYZ'
```

Falls das Dataset ein PDS-Member ist, geben Sie den PDS-Member-Namen (1 bis 8 Stellen) in Klammern hinter dem Dataset-Namen (1 bis 44 Stellen) an. Zum Beispiel:

```
DEFINE PRINTER (4) OUTPUT 'TEST.PRINT.PDS(TEST1)'
```

Falls das angegebene Member nicht existiert, wird ein neues Member unter diesem Namen angelegt.

HFS-Datei

Zum Beispiel:

```
DEFINE PRINTER (14) OUTPUT '/u/nat/rec/test.txt'
```

Der angegebene Pfadname muß existieren. Wenn das DEFINE PRINTER-Statement ausgeführt wird, wird die HFS-Datei dynamisch zugewiesen. Falls das angegebene Member nicht existiert, wird ein neues Member unter diesem Namen angelegt.

Bei dynamischer Zuweisung des Datasets werden folgende OS/390-Pfadoptionen verwendet:

```
PATHOPTS=(OCREAT,OTRUNC,ORDWR)
PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)
FILEDATA=TEXT
```

Wird eine HFS-Datei geschlossen, wird sie automatisch von OS/390 freigegeben (unabhängig vom Wert des Subparameters FREE im NTPRINT-Makro bzw. PRINT-Profilparameter).

JES-Spoolfile-Klasse

Um ein JES-Spool-Dataset zu erzeugen, geben Sie `SYSOUT=x` an (wobei *x* die gewünschte Spoolfile-Klasse ist). Für die Standard-Spoolfile-Klasse geben Sie `SYSOUT=*` an.

Beispiele:

```
DEFINE PRINTER (10) OUTPUT 'SYSOUT=A'
DEFINE PRINTER (12) OUTPUT 'SYSOUT=*'
```

Um zusätzliche Parameter für die dynamische Zuweisung anzugeben, verwenden Sie statt des `DEFINE PRINTER`-Statements den User Exit `USR2021` in Library `SYSEXT`.

NULLFILE

Um ein Dummy-Dataset zuzuweisen, geben Sie `NULLFILE` als *operand1* an:

```
DEFINE PRINTER (n) OUTPUT 'NULLFILE'
```

Dies entspricht der JCL-Definition:

```
// DD-name DD DUMMY
```

Zuweisung und Freigabe von Datasets

Wenn das `DEFINE PRINTER`-Statement ausgeführt wird und ein physischer Dataset-Name, eine HFS-Datei, eine Spoolfile-Klasse oder ein Dummy-Dataset angegeben wurde, wird das entsprechende Dataset dynamisch zugewiesen. Wenn eine logische Druckdatei bereits geöffnet ist, wird sie automatisch geschlossen, außer wenn der Profilparameter `CLOSE=FIN` angegeben wurde, wobei dann ein Fehler ausgegeben wird.

Ein bestehendes Dataset mit dem gleichen aktuellen DD-Namen wird automatisch freigegeben, bevor das neue Dataset zugewiesen wird. Druckdateien, die dynamisch zugewiesen werden sollen, müssen im Natural-Parametermodul mit `AM=STD` definiert werden.

Um Fehler durch verfrühtes Öffnen von beim Programmstart noch nicht zugewiesenen Druckdateien zu vermeiden, sollten Druckdateien mit dem Subparameter `OPEN=ACC` (Öffnen bei erstem Zugriff) im `NTPRINT`-Makro bzw. `PRINT`-Profilparameter definiert werden.

Im Falle einer HFS-Datei oder einer im `NTPRINT`-Makro bzw. `PRINT`-Profilparameter mit Subparameter `FREE=ON` definierten Druckdatei wird die Druckdatei automatisch freigegeben, sobald sie geschlossen worden ist.

Als Alternative für die dynamische Zuweisung und Freigabe von Datasets steht Ihnen der User Exit `USR2021` in Library `SYSEXT` zur Verfügung. Dieser User Exit ermöglicht auch die Angabe zusätzlicher Parameter für die dynamische Zuweisung.

Druckdateien in Server-Umgebungen

In Server-Umgebungen kann es zu Fehlern kommen, wenn mehrere Natural-Sessions versuchen, ein Dataset mit dem gleichen DD-Namen zuzuweisen oder zu öffnen. Um dies zu vermeiden, geben Sie entweder im NTPRINT-Makro bzw. PRINT-Profilparameter den Subparameter DEST=* an oder Sie geben im DEFINE PRINTER-Statement OUTPUT '*' an; Natural generiert dann einen eindeutigen DD-Namen bei der Zuweisung der physischen Datasets, wenn das erste DEFINE PRINTER-Statement für die betreffende Druckdatei ausgeführt wird.

Alle Druckdateien, deren DD-Namen mit "CM" anfangen, werden von allen Sessions in einer Server-Umgebung gemeinsam benutzt. Eine solche Druckdatei wird von der ersten Session geöffnet, aber erst bei Beendigung des Servers physisch geschlossen. Weitere Informationen siehe Abschnitt **Natural as a Server** in der *Natural Operations Documentation for Mainframes*.

Drucker unter VM/CMS mit Zugriffsmethode AM=STD (Standard-Batch)

Unter VM/CMS können Sie für einen Drucker, der (entweder automatisch in der JCL oder im NTPRINT-Makro des Natural-Parametermoduls bzw. mit dem dynamischen Profilparameter PRINT) mit Zugriffsmethode AM=STD definiert ist, mit *operand1* einen logischen oder einen physischen Dataset-Namen angeben, der dieser Druckernummer zugewiesen werden soll.

In diesem Fall gilt dasselbe wie unter OS/390 (siehe Abschnitt **Drucker unter OS/390 mit Zugriffsmethode AM=STD (Standard-Batch)**), aber mit folgenden Unterschieden:

- Anstatt der dynamischen Zuweisung über MVS SVC 99 wird das CMS-Kommando FILEDEF verwendet, um ein Dataset zu definieren.
- HFS-Dateien werden nicht unterstützt.
- JES Spool-Klassen werden nicht unterstützt.
- Außerdem wird die folgende Syntax verwendet:

```
DEFINE PRINTER (n) OUTPUT ('fname ftype fmode (options)')
```

Dies führt zur Generierung des CMS-Kommandos:

```
FILEDEF ddname-n DISK fname ftype fmode (options)
```

- Weiterhin ist die folgende Syntax zulässig:

```
DEFINE PRINTER (n) OUTPUT ('FILEDEF=filedef-parameters')
```

Dies führt zur Generierung des CMS-Kommandos:

```
FILEDEF ddname-n = filedef-parameters.
```

Drucker unter BS2000/OSD mit Zugriffsmethode AM=STD (Standard-Batch)

Unter BS2000/OSD können Sie für einen Drucker, der (entweder automatisch in der JCL oder im NTPRINT-Makro des Natural-Parametermoduls bzw. mit dem dynamischen Profilparameter PRINT) mit Zugriffsmethode AM=STD definiert ist, mit *operand1* einen Dateinamen, Link-Namen oder eine Systemdatei angeben, der/die dieser Druckernummer zugewiesen werden soll.

In diesem Fall kann *operand1* 1 bis 253 Stellen lang sein und eins der folgenden sein:

- ein BS2000/OSD Link-Name (1 bis 8 Stellen)
- ein BS2000/OSD Dateiname (9 bis 54 Stellen)
- ein generischer BS2000/OSD Dateiname (Wildcard)
- ein BS2000/OSD Dateiname und Link-Name
- ein generischer BS2000/OSD Dateiname und Link-Name (Wildcard)
- die logische BS2000/OSD Systemdatei **SYSOUT**
- die logische BS2000/OSD Systemdatei **SYSLST** (oder **SYSLSTnn**, nn=01–99)
- die logische BS2000/OSD Systemdatei **SYSLST (SYSLSTnn)** mit Zuweisung zu einem Dateinamen
- die logische BS2000/OSD Systemdatei **SYSLST (SYSLSTnn)** mit Zuweisung zu einem generischen Dateinamen (Wildcard)
- *DUMMY.

Es gelten die folgenden Regeln:

- 1 Datei- und Link-Name können als Positions-Parameter oder Schlüsselwort-Parameter angegeben werden. Die entsprechenden Schlüsselwörter sind **FILE=** und **LINK=**. Das Vertauschen von Positions-Parametern und Schlüsselwort-Parametern ist zulässig aber nicht empfehlenswert.

- ② Eine Zeichenkette mit einer Länge von 1 bis 8 Stellen ohne Kommas wird als ein Link-Name interpretiert. Diese Notation ist mit früheren Versionen von Natural kompatibel.

Beispiel:

```
DEFINE PRINTER (1) OUTPUT 'P01'
```

Die entsprechende Definition mit einem Schlüsselwort-Parameter lautet wie folgt:

```
DEFINE PRINTER (1) OUTPUT 'LINK=P01'
```

- ③ Eine Zeichenkette von 9 bis 54 Stellen ohne Kommas wird als ein Dateiname interpretiert.

Beispiel:

```
DEFINE PRINTER (2) OUTPUT 'NATURAL31.TEST.PRINTER02'
```

Die entsprechende Definition mit einem Schlüsselwort-Parameter lautet wie folgt:

```
DEFINE PRINTER (2) OUTPUT 'FILE=NATURAL31.TEST.PRINTER02'
```

- ④ Die folgende Eingabe wird ohne Berücksichtigung der Länge interpretiert und bildet deshalb eine Ausnahme zu den Regeln 2 und 3:

- Eingabe Schlüsselwort: LINK=, FILE=
- *DUMMY
- NULLFILE (entspricht *DUMMY)
- *
- *.*
- SYSOUT
- SYSLST oder SYSLST(*nn*)

Beispiel: DEFINE PRINTER (7) OUTPUT 'FILE=Y' ist eine gültige Dateizuweisung und kein Link-Name, obwohl die Zeichenkette weniger als 9 Zeichen enthält.

- ⑤ Generische Dateinamen sehen wie folgt aus:

```
pnn.userid.tsn.date.time.number
```

wobei

nn eine Reportnummer

userid eine Natural User-ID mit 8 Zeichen

tsn die BS2000/OSD TSN der aktuellen Task mit 4 Ziffern

date DDMMYYYY

time HHHSS

number eine Reihenfolgennummer mit 5 Ziffern

ist.

- ⑥ Generische Link-Namen sehen wie folgt aus:

```
NPFnnnnn
```

wobei *nnnnn* eine 5-stellige Zahl ist, die nach jeder Generierung eines dynamischen Link-Namens jeweils um eins erhöht wird.

- ⑦ Das Ändern der Dateizuweisung für eine Druckernummer führt zu einem impliziten CLOSE der bis dahin zugewiesenen Druckdatei.

In allen Fällen empfiehlt es sich sehr, mit Schlüsselwort-Parametern zu arbeiten, außer wenn Sie nur einen Link-Namen (zum Beispiel: P01) angeben. Dies hilft Konflikte mit Namen zusätzlicher Reports zu vermeiden und ist für Dateinamen mit weniger als 9 Zeichen von Bedeutung.

Beispiele:

```
DEFINE PRINTER (1) OUTPUT 'LINK=SOURCE'
DEFINE PRINTER (1) OUTPUT 'FILE=SOURCE'
DEFINE PRINTER (1) OUTPUT 'SOURCE'
```

Link-Name

Beispiel:

```
DEFINE PRINTER (1) OUTPUT 'LINKP01'
```

bedeutet dasselbe wie

```
DEFINE PRINTER (1) OUTPUT 'LINK=LINKP01'
```

Eine Datei mit dem LINK 'LINKP01' muß zur Laufzeit vorhanden sein. Diese kann entweder mittels einer JCL vor dem Start von Natural oder durch dynamische Zuweisung von der aktuellen Anwendung erstellt werden. Für eine dynamische Zuweisung kann der User-Exit USR2029 in der Library SYSEXT verwendet werden.

Wenn der Link vor der Ausführung als ein Ausgabemedium auf eine andere Datei, zum Beispiel 'P01', aktiv war, wird diese freigegeben oder zurückbehalten in Abhängigkeit vom Wert des Profilparameters FREE (mögliche Werte sind ON und OFF). Die Freigabe erfolgt über einen expliziten RELEASE-Aufruf an den BS2000/OSD Kommando-Prozessor.

Dateiname

Beispiel:

```
DEFINE PRINTER (2) OUTPUT 'NATURAL31.TEST.PRINTER02'
```

bedeutet dasselbe wie

```
DEFINE PRINTER (2) OUTPUT 'FILE=NATURAL31.TEST.PRINTER02'
```

Die in *operand1* angegebene Datei wird mittels eines Datei-Makroaufrufs eingerichtet und erbt den Link-Namen, der vor der Ausführung des DEFINE PRINTER-Statements für die entsprechende Druckdatei gültig war.

Generischer Dateiname

Beispiel:

```
DEFINE PRINTER (21) OUTPUT '*'
```

bedeutet dasselbe wie

```
DEFINE PRINTER (21) OUTPUT 'FILE=*'
```

Eine Datei mit einem nach Regel 4 erstellten Namen wird mittels eines FILE-Makroaufrufs eingerichtet und erbt den Link-Namen, der vor Ausführung des DEFINE PRINTER-Statements für die entsprechende Druckdatei gültig war.

```
DEFINE PRINTER (22) OUTPUT 'FILE=*', LINK=GENFLK22'
```

Eine Datei mit einem nach Regel 4 erstellten Namen wird mittels eines FILE-Makroaufrufs mit angegebenem Link-Namen eingerichtet.

Dateiname und Link-Name

Beispiel:

```
DEFINE PRINTER (11) OUTPUT 'NATURAL31.TEST.PRINTER11,LNKP11'
```

bedeutet dasselbe wie:

```
DEFINE PRINTER (11) OUTPUT 'FILE=NATURAL31.TEST.PRINTER11,LINK=LNKP11'
```

was dasselbe bedeutet wie:

```
DEFINE PRINTER (11) OUTPUT 'FILE=NATURAL31.TEST.PRINTER11,LNKP11'
```

Die in *operand1* angegebene Datei wird mittels eines FILE-Makroaufrufs mit dem angegebenen Link-Namen eingerichtet und der entsprechenden Druckernummer zugewiesen.

Generischer Dateiname und Link-Name

Beispiel:

```
DEFINE PRINTER (27) OUTPUT '*,*'
```

bedeutet dasselbe wie

```
DEFINE PRINTER (27) OUTPUT 'FILE=*,LINK=*'
```

Eine Datei mit einem nach Regel 4 und 5 erstellten Dateinamen und Link-Namen wird mittels eines FILE-Makroaufrufs eingerichtet und wird der angegebenen Druckernummer (27) zugewiesen.

Anmerkung:

Wenn Datei- und Link-Name angegeben werden, wird der vorige Link-Name nicht freigegeben, ungeachtet des Wertes des Profilparameters FREE.

Systemdatei SYSOUT

Beispiel:

```
DEFINE PRINTER (14) OUTPUT 'SYSOUT'
```

Report 14 wird auf SYSOUT geschrieben.

Anmerkung:

Unter TIAM wird SYSOUT standardmäßig auf dem Bildschirm ausgegeben.

Systemdatei SYSLST

Beispiel:

```
DEFINE PRINTER (15) OUTPUT 'SYSLST'
```

Report 15 wird auf die Systemdatei SYSLST geschrieben.

Systemdatei SYSLSTnn – nn=01,...,99

Beispiel:

```
DEFINE PRINTER (16) OUTPUT 'SYSLST16'
```

Report 16 wird auf die Systemdatei SYSLST16 geschrieben.

Systemdatei SYSLST – nn – mit impliziter Zuweisung zu einer BS2000/OSD-Datei

Beispiele:

```
DEFINE PRINTER (11) OUTPUT 'SYSLST=LST.PRINTER11'
```

Die Systemdatei SYSLST wird der Datei LST.PRINTER11 zugewiesen; Report 11 wird auf die Systemdatei SYSLST geschrieben.

```
DEFINE PRINTER (13) OUTPUT 'SYSLST13=LST.PRINTER13'
```

Die Systemdatei SYSLST13 wird der Datei LST.PRINTER13 zugewiesen; Report 13 wird auf die Systemdatei SYSLST13 geschrieben.

```
DEFINE PRINTER (19) OUTPUT 'SYSLST19=*'
```

Die Systemdatei SYSLST19 wird einer Datei mit einem nach Regel 4 generierten Namen zugewiesen; Report 19 wird auf die Systemdatei SYSLST19 geschrieben.

Drucker unter Com-plete

Wird Natural unter Com-plete eingesetzt, können Sie einen nicht definierten Druckernamen angeben, auch wenn er noch nicht für Natural definiert ist.

Zuweisungsalgorithmus auf Großrechnern

Um den Namen des OUTPUT-Ausgabemediums einer Reportnummer zuzuweisen, wird ein Algorithmus ausgeführt, der für alle Zugriffsmethoden (wie mit dem Parameter *AM=xxx* des NTPRINT-Makros definiert) gleich funktioniert.

Dieser Algorithmus durchsucht die Liste der für Natural definierten Drucker (wie vom Bildschirm **SYSFILE Print File Information** angezeigt), um einen Namen zu finden, der mit dem Namen des OUTPUT-Ausgabemediums übereinstimmt. Bei dieser Suche wird die Zugriffsmethode des damit verbundenen logischen Druckers **nicht** mit berücksichtigt.

Wenn ein passender Name gefunden wird, wird der logische Drucker dieses Ausgabemediums verwendet, um den Spool-Report auszugeben. Allerdings wird die SYSFILE-Ausgabe nicht geändert, d.h. dieses Routing ist nur intern und bleibt dem Benutzer verborgen.

Wenn ein passender Name **nicht** gefunden wird, wird der logische Drucker des LETZTEN Eintrages in der Liste der definierten Drucker verwendet, um den Spool-Report auszugeben. In diesem Fall wird der logische Druckernamen physisch überschrieben. Die SYSFILE-Ausgabe reflektiert diese Änderung.

Drucker unter Natural Advanced Facilities

Benutzer von Natural Advanced Facilities können den logischen Namen jedes vordefinierten logischen Druckerprofils angeben. Dieses logische Druckerprofil muß nicht zu dem gerade aktiven Benutzerprofil gehören; es darf jedes in der NATSPOOL-Datei definierte logische Druckerprofil sein. Dieses Profil gilt nur während der Ausführung des Programms, das das DEFINE PRINTER-Statement enthält. Weitere Informationen siehe *Natural Advanced Facilities Documentation*.

Zusätzliche Reports

Mit den folgenden Namen können Sie zusätzliche Standard-Reports zuweisen:

SOURCE	Ausgabe in den Arbeitsbereich des Natural-Editors.
CONNECT	Ausgabe in ein Con-nect-Fach (nur auf Großrechnern). <i>Anmerkung für die Natural-Installation:</i> <i>das NATPCNT-Modul von Natural muss mit dem Natural-Nukleus verbunden werden.</i>
DUMMY	Ausgabe wird gelöscht.
HARDCOPY	Ausgabe an das aktuelle Hardcopy-Gerät (nur auf Großrechnern).
INFOLINE	Ausgabe in der Natural-Infoline. Näheres zur Infoline siehe Terminalkommando %X im <i>Natural Referenzhandbuch</i> .
WORKPOOL	Ausgabe in den Natural-ISPf-Workpool (nur auf Großrechnern).
CCONTROL	Nur in Großrechner-Umgebungen: CCONTROL ist der Name einer bestimmten Druckersteuerzeichen-Tabelle, die mit dem Drucker "n-1" in Verbindung steht; sie darf nicht geändert werden. Weitere Informationen entnehmen Sie dem Abschnitt Printer-Advance Control Characters im Unterabschnitt Miscellaneous – Input/Output Devices in der <i>Natural Operations for Mainframe Documentation</i> .

PROFILE/FORMS/NAME/DISP/CLASS/COPIES/PRTY

Anmerkung:

Die Klauseln FORMS, NAME, DISP, CLASS, COPIES und PRTY können nur auf Großrechnern verwendet werden.

Mit diesen Klauseln können Sie Drucker-Steuerinformationen angeben, die vom Spooling-System des TP-Monitors bzw. Betriebssystems interpretiert werden.

Sie können eine oder mehrere Klauseln angeben, aber jede nur jeweils einmal.

Mit der PROFILE-Klausel geben Sie den Namen von einer Druckersteuerzeichen-Tabelle als *operand2* an. Eine solche Tabelle wird in der Konfigurationsdatei NATCONF.CFG, bzw. für Großrechner im NTCCTAB-Makro, definiert (wie in Ihrer *Natural Installation or Operations Documentation* beschrieben).

Anmerkung:

Mit Natural Advanced Facilities können Sie die Tabelle (NTCC) online pflegen (wie in der Natural Advanced Facilities Documentation beschrieben).

Mit den anderen Klauseln können Sie Parameterwerte an das Spooling-System des TP-Monitors übergeben:

FORMS	Formular
NAME	Listname
DISP	Disposition
CLASS	Spool-Klasse
COPIES	Anzahl der Kopien
PRTY	Listing-Priorität (1 – 255)

Bei diesen Klauseln werden nur die Standardwerte für die erste Ausführung benutzt.

Wenn eine der oben aufgeführten Klauseln einmal für eine bestimmte Ausgabe definiert wurde, benutzt ein nachfolgendes DEFINE PRINTER-Statement diese Definition mit derselben Ausgabe, aber ohne diese Klausel. Wenn die vorhergehenden Definitionen in einer Natural-Umgebung nicht deutlich sind, empfiehlt die Software AG, sie in jedem einzelnen Modul mittels des DEFINE PRINTER-Statements zu setzen.

Bei den Klauseln PROFILE, FORMS und NAME darf *operand2* maximal 8 Stellen lang sein, bei der DISP-Klausel 4 Stellen, bei der CLASS-Klausel 1 Stelle.

Bei der DISP-Klausel sind die möglichen Werte für *operand2* “DEL”, “HOLD”, “KEEP” und “LEAV”. Wenn Sie die DISP-Klausel weglassen (oder falsch angeben), gilt standardmäßig “DEL”.

Operand3 und *operand4* müssen Ganzzahlen sein.

Standardwerte können mit den entsprechenden Subparametern des Profilparameters PRINT gesetzt werden.

Beispiel 1

```
DEFINE PRINTER (1) OUTPUT 'TID100'  
WRITE (1) 'PRINTED ON PRINTER TID100'  
END
```

Beispiel 2

```
DEFINE PRINTER (REPORT1 = 1) OUTPUT 'LPT1'  
WRITE (REPORT1) 'REPORT1 PRINTED ON PRINTER LPT1'  
END
```

Beispiel 3

```
DEFINE PRINTER (REPORT1 = 1) /* NO 'OUTPUT'  
WRITE (REPORT1) 'DEPENDS ON NATPARM SETTING OR JCL IN BATCH'  
                'OR ''PRINTER PARAMETER'' UNDER COM-LETE OR A/F'  
END
```

Beispiel 4

```
/* EXAMPLE 'DPIEX1': DEFINE PRINTER INFOLINE
*
SET CONTROL 'XT' /* INFOLINE TOP
SET CONTROL 'XI' /* SWITCH INFOLINE MODE
DEFINE PRINTER (1) OUTPUT 'INFOLINE'
WRITE (1) 'EXECUTING' *PROGRAM 'BY' *INIT-USER
WRITE 'TEST OUTPUT'
SET CONTROL 'XI' /* SWITCH BACK TO NORMAL
END
```

Page 1

97-06-18 13:31:25

TEST OUTPUT

IO=814,AI =650,L=0 C= ,LS=80,P =3,PLS=80,PCS=24,FLD=90,CLS=5,ADA=22

DEFINE SUBROUTINE

```
DEFINE [SUBROUTINE] subroutine-name
    statement...
{ END-SUBROUTINE
  RETURN (nur im Reporting Mode) }
```

Verwandte Statements

PERFORM, DEFINE DATA PARAMETER.

Funktion

Das Statement DEFINE SUBROUTINE dient dazu, eine Natural-Subroutine zu definieren. Aufgerufen wird eine Subroutine mit einem PERFORM-Statement.

Interne und externe Subroutinen

Eine Subroutine kann entweder innerhalb des Natural-Objekts definiert werden, das das sie aufrufende PERFORM-Statement enthält (interne Subroutine); oder sie kann in einem anderen Natural-Objekt definiert werden als dem, welches das aufrufende PERFORM-Statement enthält (externe Subroutine). Eine interne Subroutine kann entweder vor oder nach dem ersten PERFORM-Statement, mit dem sie aufgerufen wird, definiert werden.

Anmerkung:

Die Verwendung externer Subroutinen empfiehlt sich zwar, um eine klar gegliederte Anwendungsstruktur zu erhalten; allerdings verursachen externe Subroutinen einen Verarbeitungsmehraufwand. Daher sollten nur größere funktionale Blöcke in externen Subroutinen untergebracht werden.

subroutine-name

Für den Namen einer Subroutine (maximal 32 Zeichen lang) gelten die gleichen Namenskonventionen wie für Benutzervariablen (siehe *Natural Referenzhandbuch*). Der Name einer Subroutine ist unabhängig vom Namen des Moduls, in dem sie definiert ist (beide Namen können, müssen aber nicht, gleich sein).

Beenden einer Subroutine

Die Definition einer Subroutine wird mit END-SUBROUTINE beendet. Im *Reporting Mode* darf eine Subroutine auch mit RETURN beendet werden.

Einschränkungen

Eine in einer Subroutine initiierte Verarbeitungsschleife muß vor dem END-SUBROUTINE-Statement wieder geschlossen werden.

Eine interne Subroutine darf ihrerseits kein weiteres DEFINE SUBROUTINE-Statement enthalten (siehe Beispiel 1 unten).

Eine externe Subroutine (d.h. ein Objekt vom Typ Subroutine) darf nicht mehr als einen DEFINE SUBROUTINE-Statement-Block enthalten (siehe Beispiel 2 unten). Ein externer DEFINE SUBROUTINE-Block darf jedoch seinerseits interne Subroutinen enthalten (siehe Beispiel 1 unten).

Beispiel 1:

Die folgende Konstruktion ist in einem Objekt vom Typ Subroutine möglich, aber nicht in einem anderen Objekt (wo SUBR01 als interne Subroutine gälte):

```
...
DEFINE SUBROUTINE SUBR01
  ...
  PERFORM SUBROUTINE SUBR02
  PERFORM SUBROUTINE SUBR03
  ...
  DEFINE SUBROUTINE SUBR02
  /* inline subroutine...
  END-SUBROUTINE
  ...
  DEFINE SUBROUTINE SUBR03
  /* inline subroutine...
  END-SUBROUTINE
END-SUBROUTINE
END
```

Beispiel 2 (ungültig):

Die folgende Konstruktion ist in einem Objekt vom Typ Subroutine *nicht* erlaubt:

```
...  
DEFINE SUBROUTINE SUBR01  
...  
END-SUBROUTINE  
DEFINE SUBROUTINE SUBR02  
...  
END-SUBROUTINE  
END
```

Welche Daten einer Subroutine zur Verfügung stehen

Interne Subroutinen

An eine interne Subroutine können mit dem PERFORM-Statement keine Parameter vom aufrufenden Programm übergeben werden.

Eine interne Subroutine kann auf die aktuelle Global Data Area sowie die vom aufrufenden Programm verwendete Local Data Area zugreifen.

Externe Subroutinen

Eine externe Subroutine kann auf die aktuelle Global Data Area zugreifen. Außerdem können Sie mit dem PERFORM-Statement Parameter direkt vom aufrufenden Objekt an die externe Subroutine übergeben; dadurch können Sie die Größe der Global Data Area klein halten.

Eine externe Subroutine kann nicht auf die im aufrufenden Programm definierte Local Data Area zugreifen; allerdings kann eine externe Subroutine eine eigene Local Data Area haben.

Beispiel 1

```

/* EXAMPLE 'DSREX1S': DEFINE SUBROUTINE (STRUCTURED MODE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE (A20/2)
  2 PHONE
1 #ARRAY (A75/1:4)
1 REDEFINE #ARRAY
  2 #ALINE (A25/1:4,1:3)
1 #X (N2) INIT <1>
1 #Y (N2) INIT <1>
END-DEFINE
/*****
FORMAT PS=20
LIMIT 5
FIND EMPLOY-VIEW WITH NAME = 'SMITH'
MOVE NAME TO #ALINE (#X,#Y)
MOVE ADDRESS-LINE(1) TO #ALINE (#X+1,#Y)
MOVE ADDRESS-LINE(2) TO #ALINE (#X+2,#Y)
MOVE PHONE TO #ALINE (#X+3,#Y)
IF #Y = 3
  RESET INITIAL #Y
  PERFORM PRINT
ELSE
  ADD 1 TO #Y
END-IF
AT END OF DATA
  PERFORM PRINT
END-ENDDATA
END-FIND
/*****
DEFINE SUBROUTINE PRINT
WRITE NOTITLE (AD=OI) #ARRAY(*)
RESET #ARRAY(*)
SKIP 1
END-SUBROUTINE
/*****
END

```

SMITH ENGLANDSVEJ 222 554349	SMITH 3152 SHETLAND ROAD MILWAUKEE (414)877-4563	SMITH 14100 ESWORTHY RD. MONTERREY (408)994-2260
SMITH 5 HAWTHORN OAK BROOK (312)150-9351	SMITH 2307 DARIUS LANE TAMPA (813)131-4010	

Beispiel 2

```

/* EXAMPLE 'DSREX2'
/* SAMPLE STRUCTURE FOR EXTERNAL SUBROUTINE USING GLOBAL DATA
/* *****
/* PROGRAM CONTAINING SUBROUTINE
DEFINE DATA GLOBAL USING GLOBAL-1
    LOCAL 1 FIELD (N7)
END-DEFINE
/* ...
/* ...
/* ...
/* *****
/* SUBROUTINE 'SUBROUT1'
DEFINE SUBROUTINE SUBROUT1
/* ...
WRITE 'IN SUBROUTINE:' FIELD
/* ...
END-SUBROUTINE
/* *****
END

```

DEFINE WINDOW

DEFINE WINDOW <i>window-name</i>				
$\left[\text{SIZE} \left\{ \begin{array}{c} \text{AUTO} \\ \text{QUARTER} \\ \text{operand1 * operand2} \end{array} \right\} \right]$				
$\left[\text{BASE} \left\{ \begin{array}{c} \text{CURSOR} \\ \left\{ \begin{array}{c} \text{TOP} \\ \text{BOTTOM} \end{array} \right\} \left\{ \begin{array}{c} \text{LEFT} \\ \text{RIGHT} \end{array} \right\} \\ \text{operand3 / operand4} \end{array} \right\} \right]$				
$\left[\text{REVERSED} \left[(\text{CD}=\text{background-color}) \right] \right]$				
$\left[\text{TITLE } \text{operand5} \right]$				
$\left[\text{CONTROL} \left\{ \begin{array}{c} \text{WINDOW} \\ \text{SCREEN} \end{array} \right\} \right]$				
$\left[\text{FRAMED} \left\{ \begin{array}{c} \left[\text{ON} \right] \left[(\text{CD}=\text{frame-color}) \right] \left[\text{position-clause} \right] \\ \text{OFF} \end{array} \right\} \right]$				

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	N P I	ja	nein
Operand2	C S	N P I	ja	nein
Operand3	C S	N P I	ja	nein
Operand4	C S	N P I	ja	nein
Operand5	C S	A	ja	nein

Verwandte Statements

SET WINDOW, INPUT WINDOW='window-name', SET CONTROL 'W'.

Funktion

Das DEFINE WINDOW-Statement dient dazu, die Größe, Position und Attribute eines Bildschirmfensters (“Window”) zu definieren.

Ein Fenster ist der Ausschnitt einer von einem Programm erzeugten logischen Seite, der auf dem Bildschirm zu sehen ist. Das Fenster ist ständig vorhanden, auch wenn Sie sich dessen nicht bewußt sind, da die Größe des Fensters, solange Sie sie nicht anders definieren, mit der Größe Ihres Bildschirms identisch ist.

Mit einem DEFINE WINDOW-Statement wird ein Fenster nicht aktiviert; dies geschieht mit einem SET WINDOW-Statement oder der WINDOW-Klausel eines INPUT-Statements.

Anmerkung:

Es gibt stets nur ein Natural-Fenster, und zwar das jeweils neueste. Vorherige Fenster mögen auf dem Bildschirm noch sichtbar sein, sind aber nicht länger aktiv und werden von Natural ignoriert. Sie können Eingaben nur im jeweils neuesten Fenster machen. Sollte der Platz hierzu nicht ausreichen, müssen Sie das Fenster vorher entsprechend vergrößern.

Kontrolle über den ganzen Schirm

Auch wenn ein Fenster aktiv ist, behält Natural die Kontrolle über den gesamten Bildschirm. Dies hat für Einzel-Session-Systeme wie CICS oder TSO keine Bedeutung; wenn Natural jedoch unter einem Mehr-Session-System wie Com-plete oder Multi-pass läuft, wird der gesamte Natural-Schirm — d.h. nicht nur das gerade aktive Fenster sondern auch der Natural-Schirm “unter” dem Fenster — angezeigt, wenn eine vorübergehend unterbrochene Session wiederaufgenommen wird; gleichzeitig werden die Attribute der Felder auf dem ganzen Schirm, die von dem Fenster teilweise überlagert werden, durch das Fenster nicht beeinflusst.

window-name

Der *window-name* identifiziert das Fenster. Der Name darf bis zu 32 Stellen lang sein. Für Fensternamen gelten die gleichen Namenskonventionen wie für Benutzervariablen (siehe *Natural Referenzhandbuch*).

SIZE

Mit der SIZE-Klausel bestimmen Sie die Größe des Fensters.

- **SIZE AUTO** — Die Größe des Fensters wird von Natural automatisch zur Laufzeit festgelegt. Die Größe wird wie folgt durch die in das Fenster hineingenerierten Daten bestimmt:

Die Zeilenanzahl des Fensters entspricht der Anzahl der generierten INPUT-Zeilen (plus gegebenenfalls der PF-Tastenleiste, der Meldungszeile und der Statistikzeile/Infoline).

Die Spaltenanzahl des Fensters wird durch die längste INPUT-Zeile bestimmt: Natural sucht, ausgehend von den Zeilenenden, nach dem signifikanten Byte, das am weitesten rechts in einer Zeile steht. Daher kann es vorkommen, daß reine Eingabefelder oder modifizierbare Felder (AD=A bzw. AD=M) abgeschnitten werden; um dies zu verhindern, können Sie entweder einen einbuchstabigen Text hinter so ein Feld stellen oder die gewünschte Fenstergröße explizit mit "SIZE operand1 * operand2" angeben.

Anmerkung: Der Titel ist nicht Bestandteil der Fensterdaten. Deshalb wird er abgeschnitten, wenn die Fenstergröße festgelegt worden ist, wie oben beschrieben, und der Titel länger als das Fenster ist.

- **SIZE QUARTER** — Die Größe des Fensters entspricht einem Viertel der physischen Bildschirmgröße.
- **SIZE operand1 * operand2** — Die Größe des Fensters beträgt n Zeilen mal n Spalten. *Operand1* bestimmt die Anzahl der Zeilen, *operand2* die Anzahl der Spalten. Die beiden Operanden dürfen keine Dezimalstellen enthalten.

Ist das Fenster gerahmt (FRAMED), so schließt die angegebene Größe den Rahmen mit ein.

Die kleinstmögliche Fenstergröße ist:

- ohne Rahmen: 2 Zeilen mal 10 Spalten,
- mit Rahmen: 4 Zeilen mal 13 Spalten.

Die größtmögliche Fenstergröße ist die Größe des physischen Bildschirms.

Wenn Sie die SIZE-Klausel weglassen, gilt standardmäßig SIZE AUTO.

Anmerkung:

Auf Großrechnern benötigt Natural zusätzliche Spalten für sogenannte Attribut-Bytes, um Daten auf dem Bildschirm anzeigen zu können (auf anderen Plattformen sind solche Attribut-Bytes nicht erforderlich). Dadurch ist auf einem Großrechner der von einem Fenster überlagerte Bildschirmbereich breiter und der innerhalb eines Fensters sichtbare Seitenausschnitt schmaler als auf anderen Plattformen.

*Beispiel: Angenommen, die Größe eines Fensters ist mit "SIZE 5 * 15" definiert (d.h. mit einer Breite von 15 Spalten):*

- Auf Großrechnern ist der von dem Fenster überlagerte Bildschirmbereich 16 Spalten breit; der innerhalb des Fensters sichtbare Seitenausschnitt ist 14 Spalten breit ohne Rahmen bzw. 10 Spalten mit Rahmen.*
- Auf anderen Plattformen ist der vom Fenster überlagerte Bildschirmbereich 15 Spalten breit; der innerhalb des Fensters sichtbare Seitenausschnitt ist 15 Spalten breit ohne Rahmen bzw. 13 Spalten mit Rahmen.*

BASE

Mit der BASE-Klausel bestimmen Sie die Position des Fensters auf dem physischen Bildschirm.

- **BASE CURSOR** plziert die obere linke Ecke des Fensters an die aktuelle Cursor-Position. Die Cursor-Position ist die *physische* Position des Cursors auf dem Bildschirm.
Wenn es aufgrund der Größe des Fensters nicht möglich ist, das Fenster an die Cursor-Position zu plazieren, plaziert Natural es automatisch so dicht wie möglich an die gewünschte Position.
- **BASE TOP/BOTTOM LEFT/RIGHT** plaziert das Fenster in die obere linke, untere linke, obere rechte bzw. untere rechte Ecke des Bildschirms.
- **BASE operand3/operand4** — Dies plaziert die obere linke Ecke des Fensters in die angegebene Zeile/Spalte auf dem physischen Bildschirm. *Operand3* bestimmt die Zeilennummer, *operand4* die Spaltennummer. Die beiden Operanden dürfen keine Dezimalstellen enthalten.
Wenn es aufgrund der Größe des Fensters nicht möglich ist, das Fenster an die angegebene Position zu plazieren, erhalten Sie eine Fehlermeldung.

Wenn Sie die BASE-Klausel weglassen, gilt standardmäßig BASE CURSOR.

REVERSED

REVERSED bewirkt, daß das Fenster invers angezeigt wird (falls der verwendete Bildschirm dies ermöglicht; falls nicht, wird REVERSED ignoriert).

REVERSED (CD=*background-color*)

Dies bewirkt, daß das Fenster invers und der Fensterhintergrund in der angegebenen Farbe (*background-color*) angezeigt wird (falls der verwendete Bildschirm dies ermöglicht; falls nicht, wird die betreffende Angabe ignoriert).

Informationen über gültige Farbcodes s. Session-Parameter CD im *Natural Referenzhandbuch*.

TITLE *operand5*

Mit der TITLE-Klausel können Sie eine Überschrift für das Fenster angeben. Die angegebene Überschrift (*operand5*) wird zentriert in der oberen Rahmenzeile des Fensters angezeigt. Die Überschrift kann entweder als Textkonstante (in Apostrophen) oder als Inhalt einer Benutzervariablen angegeben werden. Ist die Überschrift länger als das Fenster, wird sie abgeschnitten. Die Überschrift wird nur angezeigt, wenn das Fenster gerahmt (FRAMED) ist; wenn FRAMED OFF angegeben ist, wird die TITLE-Klausel ignoriert.

Anmerkung:

Wenn der Titel nachfolgende Leerzeichen enthält, werden diese entfernt.

Wenn das erste Zeichen des Titels ein Leerzeichen ist, wird hinter dem Titel automatisch ein Leerzeichen angehängt.

CONTROL

Mit der CONTROL-Klausel bestimmen Sie, ob die PF-Tastenleiste, die Meldungszeile und die Statistikzeile innerhalb oder außerhalb des Fensters angezeigt werden.

CONTROL WINDOW

CONTROL WINDOW zeigt die Zeilen innerhalb des Fensters an.

CONTROL SCREEN

CONTROL SCREEN zeigt die Zeilen auf dem vollen physischen Schirm außerhalb des Fensters an.

Wenn Sie die CONTROL-Klausel weglassen, gilt standardmäßig CONTROL WINDOW.

FRAMED

Standardmäßig, d.h. wenn Sie die FRAMED-Klausel weglassen, wird das Fenster mit Rahmen angezeigt. Wenn Sie FRAMED OFF angeben, wird die Rahmung und alles, was am Rahmen hängt (Fensterüberschrift und Positionsinformation), ausgeschaltet.

Die obere und die untere Rahmenlinie sind cursor-sensitiv: Sie können im Fenster vor, zurück, nach links oder rechts blättern, indem Sie einfach den Cursor auf das entsprechende Symbol (<, -, +, oder >; siehe *position-clause* unten) stellen und FREIG drücken. Wenn keine Symbole angezeigt werden, können Sie im Fenster vor- oder zurückblättern, indem Sie den Cursor in die obere (zum Zurückblättern) bzw. untere (zum Vorblättern) Rahmenlinie plazieren und FREIG drücken.

Anmerkung:

Falls das Fenster kleiner als 4 Zeilen mal 12 (bzw. 13 auf Großrechnern) Spalten ist, ist der Rahmen nicht sichtbar.

FRAMED (CD=*frame-color*)

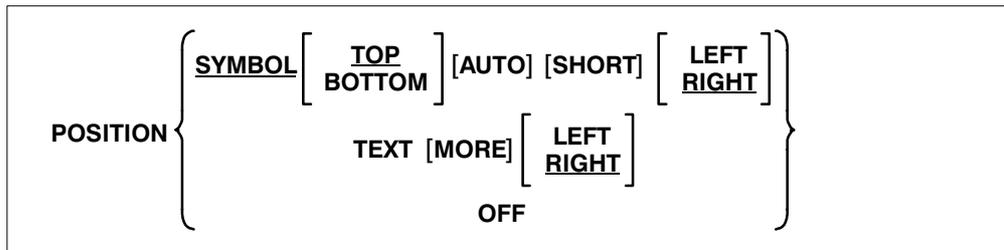
Dies bewirkt, daß der Fensterrahmen in der angegebenen Farbe (*frame-color*) angezeigt wird (falls ein Farbbildschirm verwendet wird; falls nicht, wird die Farbangabe ignoriert).

Informationen über gültige Farbcodes s. Session-Parameter CD im *Natural Referenzhandbuch*.

position-clause

Anmerkung:

Die **POSITION**-Klausel ist nur auf Großrechnern wirksam; auf allen anderen Plattformen wird sie ignoriert.



Mit der **POSITION**-Klausel steuern Sie die Anzeige von Informationen über die Position des Fensters auf der logischen Seite: Im Rahmen des Fensters wird angezeigt, in welche Richtungen die logische Seite nach oben, unten, links und rechts über das aktuelle Fenster hinausgeht. Dies gilt nur, falls die logische Seite größer als das Fenster ist; falls nicht, wird die **POSITION**-Klausel ignoriert.

Wenn Sie die **POSITION**-Klausel nicht angeben, gilt standardmäßig **POSITION SYMBOL TOP RIGHT**.

POSITION SYMBOL

POSITION SYMBOL bewirkt, daß die Positionsinformationen in Form von Symbolen angezeigt werden: “More: < – + >”. Die Informationen werden in der oberen und/oder unteren Rahmenzeile angezeigt.

TOP/BOTTOM bestimmt, ob die Positionsinformationen in der oberen oder unteren Rahmenzeile angezeigt werden.

AUTO ist nur relevant, wenn die logische Seite horizontal vollständig im Fenster sichtbar ist, d.h. wenn nur “–” und/oder “+” angezeigt werden soll. In diesem Fall schaltet **AUTO** automatisch von den Symbolen auf die Wörter “Top”, “Bottom” bzw. “More” um.

SHORT bewirkt, daß das Wort “More:” vor den Symbolen “< – + >” nicht angezeigt wird.

LEFT/RIGHT bestimmt, ob die Positionsinformationen im linken oder im rechten Teil der Rahmenzeile angezeigt werden.

POSITION TEXT

POSITION TEXT bewirkt, daß die Positionsinformationen in der oberen und/oder unteren Rahmenzeile in Textform angezeigt werden, und zwar mit den Wörtern “More”, “Top” und “Bottom”. Die Wörter sind sprachabhängig und können durch entsprechendes Setzen des Sprachcodes auch in einer anderen Sprache angezeigt werden.

POSITION TEXT MORE unterdrückt die Wörter “Top” und “Bottom” und zeigt nur das Wort “More” je nach Situation in der oberen oder unteren Rahmenzeile oder in beiden.

LEFT/RIGHT bestimmt, ob die Positionsinformationen im linken oder rechten Teil der Rahmenzeile angezeigt werden.

POSITION OFF

POSITION OFF bewirkt, daß überhaupt keine Positionsinformationen angezeigt werden.

Schutz von Eingabefeldern in einem Fenster

Für Eingabefelder (AD=A oder AD=M), die sich nicht vollständig innerhalb des Fensters befinden, gilt folgendes:

- Ein Eingabefeld, dessen Anfang außerhalb des Fensters liegt, wird immer zu einem geschützten Feld gemacht.
- Ein Eingabefeld, das im Fenster beginnt aber außerhalb des Fensters endet, wird nur dann geschützt, wenn der Wert, den es enthält, nicht vollständig innerhalb des Fensters sichtbar ist. Bitte beachten Sie, daß es hierbei darauf ankommt, ob die *Wertlänge*, nicht die *Feldlänge*, über das Fenster hinausgeht. Füllzeichen (wie mit dem Profilparameter FC angegeben) zählen nicht als Teil des Wertes.

Falls Sie in ein derart geschütztes Eingabefeld Eingaben machen möchten, müssen Sie zunächst die Fenstergröße so ändern, daß sich der Anfang des Feldes bzw. das Ende des Feldwertes innerhalb des Fensters befindet.

Aufrufen unterschiedlicher Fenster

Ein DEFINE WINDOW-Statement darf nicht innerhalb einer logischen Bedingung stehen. Wollen Sie in Abhängigkeit von einer Bedingung unterschiedliche Fenster aufrufen, verwenden Sie dazu verschiedene SET WINDOW-Statements (bzw. INPUT-Statements mit WINDOW-Klausel) in einer Bedingung.

Beispiel

```

/* EXAMPLE 'DWDEX1': DEFINE WINDOW
DEFINE DATA LOCAL
01 #I(P3)
END-DEFINE
*
SET KEY PF1='%W<<' PF2='%W>>' PF4='%W—' PF5='%W++'
*
DEFINE WINDOW WIND1
  SIZE QUARTER
  BASE TOP RIGHT
  FRAMED ON POSITION SYMBOL AUTO
*
SET WINDOW 'WIND1'
FOR #I = 1 TO 10
  WRITE 25X #I 'THIS IS SOME LONG TEXT' #I
LOOP
*
END

```

```

> r                                     +-----More:      + >+
All      ....+....1....+....2....+....3.. ! Page      1      !
0010 /* EXAMPLE 'DWDEX1': DEFINE WIND !
0020 DEFINE DATA LOCAL                !                      1 THIS !
0030 01 #I(P3)                          !                      2 THIS !
0040 END-DEFINE                          !                      3 THIS !
0050 *                                    !                      4 THIS !
0060 SET KEY PF1='%W<<' PF2='%W>>' PF !                      5 THIS !
0070 *                                    !                      6 THIS !
0080 DEFINE WINDOW WIND1                 !                      7 THIS !
0090   SIZE QUARTER                      ! MORE                  !
0100   BASE TOP RIGHT                    +-----+
0110   FRAMED ON POSITION SYMBOL AUTO
0120 *
0130 SET WINDOW 'WIND1'
0140 FOR #I = 1 TO 10
0150   WRITE 25X #I 'THIS IS SOME LONG TEXT' #I
0160 END-FOR
0170 *
0180 END
0190
0200
      ....+....1....+....2....+....3....+....4....+....5....+... S 18   L 1

```

DEFINE WORK FILE

DEFINE WORK FILE *n operand1* [**TYPE** *operand2*]

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	ja	nein
Operand2	C S	A	ja	nein

Verwandte Statements

WRITE WORK FILE, READ WORK FILE, CLOSE WORK FILE.

Funktion

Das Statement DEFINE WORK FILE dient dazu, innerhalb einer Natural-Anwendung einer Natural-Arbeitsdatei-Nummer einen Dateinamen zuzuweisen.

Damit können Sie Arbeitsdatei-Zuweisungen innerhalb Ihrer Natural-Session dynamisch vornehmen bzw. ändern sowie auf anderer Ebene gemachte Arbeitsdatei-Zuweisungen überschreiben.

Ist bei der Ausführung dieses Statements die angegebene Arbeitsdatei bereits offen, bewirkt dieses Statement implizit, daß die Arbeitsdatei geschlossen wird.

Arbeitsdatei-Nummer (*n*)

n ist die Nummer der Arbeitsdatei (1 bis 32). Dies ist die Nummer, wie sie in einem READ WORK FILE-, WRITE WORK FILE- oder CLOSE WORK FILE-Statement verwendet wird.

Auf Großrechnern muß die angegebene Arbeitsdatei mit Zugriffsmethode AM=STD definiert sein (im NETWORK-Makro des Natural-Parametermoduls bzw. mit dem dynamischen Profilparameter WORK). Für Arbeitsdateien, die mit einer anderen Zugriffsmethode definiert sind, wird das DEFINE WORK FILE-Statement ignoriert.

Arbeitsdatei-Name (*operand1*)

operand1 ist der Name der Arbeitsdatei.

Arbeitsdatei-Name auf OpenVMS, UNIX und Windows

Der Dateiname (*operand1*) darf Environment-Variablen bzw. "Logicals" enthalten.

Wenn eine Datei dieses Namens nicht existiert, wird sie erstellt.

Arbeitsdatei-Name auf Großrechnern

Als *operand1* geben Sie den Namen des Datasets an, das der Arbeitsdatei-Nummer zugewiesen werden soll.

Operand1 kann 1 bis 253 Stellen lang sein. Sie können entweder einen logischen oder einen physischen Dateinamen angeben.

Arbeitsdateityp – *operand2*

Operand2 gibt den Arbeitsdateityp an.

Operand2 kann auf Großrechnern nicht angegeben werden.

Mögliche Werte für *operand2* sind:

Arbeitsdateityp	Verwendung
DEFAULT	Dateityp von Extension für Aufwärts-Kompatibilität festlegen.
TRANSFER	Von/nach PC mit Entire Connection übertragen.
SAG	Binäres Format.
ASCII	“Text”-Dateien mit durch CR-Zeilenvorschub abgeschlossenen Datensätzen.
ASCII-COMPRESSED	Als ASCII-Dateien, aber nachstehende Leerstellen werden abgeschnitten.
ENTIRECONNECTION	Direktes Lesen/Schreiben (READ/WRITE) in Entire Connection-Format.
UNFORMATTED	Formatier-Informationen werden nicht geschrieben (weder für Felder noch für Datensätze).
PORTABLE	Dateien, die dynamische Variablen vollständig bearbeiten können und auch portiert werden können: z.B. von einer Maschine des Typs “little endian” (höherwertiges Byte vorne) auf eine Maschine des Typs “big endian” (höherwertiges Byte hinten) und umgekehrt.

Beim Wert von *operand2* wird nicht nach Groß- und Kleinschreibung unterschieden; er muß in Anführungszeichen eingeschlossen oder in einer alphanumerischen Variable angegeben werden.

Beispiele:

```
DEFINE WORK FILE 17 #FILE-TYPE 'unformatted'
#TYPE := 'SAG'
DEFINE WORK FILE 18 #FILE TYPE #TYPE
```

Arbeitsdatei-Name unter OS/390

Unter OS/390 kann für eine Arbeitsdateinummer, die (entweder automatisch in der JCL, im NETWORK-Makro des Natural-Parametermoduls bzw. dynamisch mittels des Profilparameters WORK) mit Zugriffsmethode AM=STD definiert ist, *operand1* folgendes sein:

- ein logischer Dataset-Name (DD-Name, 1 bis 8 Stellen);
- ein physischer Dataset-Name eines katalogisierten Datasets (1 bis 44 Stellen) oder ein physischer Dataset-Member-Name;
- ein Pfad- und Member-Name einer HFS-Datei (1 bis 253 Stellen) in einer MVS-UNIX-Services-Umgebung;
- eine JES-Spoolfile-Klasse;
- "NULLFILE" (bezeichnet ein Dummy-Dataset).

Logische Dataset-Namen

Zum Beispiel:

```
DEFINE WORK FILE 21 'SYSOUT'
```

Das angegebene Dataset muß zugewiesen worden sein, bevor das DEFINE WORK FILE-Statement ausgeführt wird.

Die Zuweisung kann über JCL, CLIST oder dynamische Zuweisung (SVC 99) erfolgen. Für dynamische Zuweisung können Sie den User Exit USR2021 in Library SYSEXT verwenden.

Der im DEFINE WORK FILE-Statement angegebene Dataset-Name überschreibt den im Subparameter DEST des NETWORK-Makros bzw. WORK-Profilparameters angegebenen Namen.

Optional kann dem Dataset-Namen "DDN=" vorangestellt werden, um anzuzeigen, daß es sich um einen DD-Namen handelt. Zum Beispiel:

```
DEFINE WORK FILE 22 'DDN=XYZ'
```

Physische Dataset-Namen

Zum Beispiel:

```
DEFINE WORK FILE 23 'TEST.WORK.FILE'
```

Das angegebene Dataset muß in katalogisierter Form vorhanden sein. Wenn das DEFINE WORK FILE-Statement ausgeführt wird, wird das Dataset dynamisch über SVC 99 mit dem aktuellen DD-Namen und der Option DISP=SHR zugewiesen.

Wenn der Dataset-Name 8 Stellen oder kürzer ist und keinen Punkt “.” enthält, könnte er als DD-Name mißinterpretiert werden. Um dies zu vermeiden, stellen Sie ihm “DSN=” voran. Zum Beispiel:

```
DEFINE WORK FILE 22 'DSN=WORKXYZ'
```

Falls das Dataset ein PDS-Member ist, geben Sie den PDS-Member-Namen (1 bis 8 Stellen) in Klammern hinter dem Dataset-Namen (1 bis 44 Stellen) an. Zum Beispiel:

```
DEFINE WORK FILE 4 'TEST.WORK.PDS(TEST1)'
```

Falls das angegebene Member nicht existiert, wird ein neues Member unter diesem Namen angelegt.

HFS-Dateien

Zum Beispiel:

```
DEFINE WORK FILE 14 '/u/nat/rec/test.txt'
```

Der angegebene Pfadname muß existieren. Wenn das DEFINE WORK FILE-Statement ausgeführt wird, wird die HFS-Datei dynamisch zugewiesen. Falls das angegebene Member nicht existiert, wird ein neues Member unter diesem Namen angelegt.

Bei der dynamischen Zuweisung des Datasets werden folgende OS/390-Pfadoptionen verwendet:

```
PATHOPTS=(OCREAT,OTRUNC,ORDWR)  
PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)  
FILEDATA=TEXT
```

Wird eine HFS-Datei geschlossen, wird sie automatisch von OS/390 freigegeben (unabhängig vom Wert des Subparameters FREE im NETWORK-Makro bzw. WORK-Profilparameter).

Um eine HFS-Datei zu lesen, müssen Sie statt des DEFINE WORK FILE-Statements den User Exit USR2021 verwenden, und zwar wegen der OTRUNC-Option. Diese Option setzt die HFS-Datei beim ersten Lesezugriff zurück und bewirkt eine leere Datei.

JES-Spoolfile-Klasse

Um ein JES-Spool-Dataset zu erzeugen, geben Sie `SYSOUT=x` an (wobei x die gewünschte Spoolfile-Klasse ist). Für die Standard-Spoolfile-Klasse geben Sie `SYSOUT=*` an.

Beispiele:

```
DEFINE WORK FILE 10 'SYSOUT=A'  
DEFINE WORK FILE 12 'SYSOUT=*
```

Um zusätzliche Parameter für die dynamische Zuweisung anzugeben, verwenden Sie statt des `DEFINE WORK FILE`-Statements den User Exit `USR2021` in Library `SYSEXT`.

NULLFILE

Um ein Dummy-Dataset zuzuweisen, geben Sie `NULLFILE` als *operand1* an:

```
DEFINE WORK FILE n 'NULLFILE'
```

Dies entspricht der JCL-Definition:

```
// DD-name DD DUMMY
```

Zuweisung und Freigabe von Datasets

Wenn das DEFINE WORK FILE-Statement ausgeführt wird und ein physischer Dataset-Name, eine HFS-Datei, eine Spoolfile-Klasse oder ein Dummy-Dataset angegeben wurde, wird das entsprechende Dataset automatisch zugewiesen. Wenn die logische Datei bereits geöffnet ist, wird sie automatisch geschlossen, außer wenn der Profilparameter CLOSE=FIN angegeben wurde, wobei dann eine Fehlermeldung ausgegeben wird.

Außerdem wird ein bestehendes Dataset mit dem gleichen aktuellen DD-Namen automatisch freigegeben, bevor das neue Dataset zugewiesen wird. Arbeitsdateien, die dynamisch zugewiesen werden sollen, müssen im Natural-Parametermodul mit AM=STD definiert werden.

Um Fehler durch verfrühtes Öffnen von beim Programmstart noch nicht zugewiesenen Arbeitsdateien zu vermeiden, sollten Arbeitsdateien mit dem Subparameter OPEN=ACC (Öffnen bei erstem Zugriff) im NETWORK-Makro bzw. WORK-Profilparameter definiert werden.

Im Falle einer HFS-Datei oder einer im NETWORK-Makro bzw. WORK-Profilparameter mit Subparameter FREE=ON definierten Arbeitsdatei wird die Arbeitsdatei automatisch freigegeben, sobald sie geschlossen worden ist.

Als Alternative steht Ihnen für die dynamische Zuweisung und Freigabe von Datasets der User Exit USR2021 in Library SYSEXT zur Verfügung. Dieser User Exit ermöglicht auch die Angabe zusätzlicher Parameter für die dynamische Zuweisung.

Arbeitsdateien in Server-Umgebungen

In Server-Umgebungen kann es zu Fehlern kommen, wenn mehrere Natural-Sessions versuchen, ein Dataset mit dem gleichen DD-Namen zuzuweisen oder zu öffnen. Um dies zu vermeiden, geben Sie entweder im NETWORK-Makro bzw. WORK-Profilparameter den Subparameter DEST=* an, oder Sie programmieren DEFINE WORK FILE '*' vor dem eigentlichen DEFINE WORK FILE-Statement; Natural generiert dann einen eindeutigen DD-Namen bei der Zuweisung der physischen Datasets, wenn das erste DEFINE WORK FILE-Statement für die betreffende Arbeitsdatei ausgeführt wird.

Alle Arbeitsdateien, deren DD-Namen mit "CM" anfangen, werden von allen Sessions in einer Server-Umgebung gemeinsam benutzt. Eine gemeinsame benutzte Arbeitsdatei, die für Ausgabe von der ersten Session geöffnet wird, wird bei Beendigung des Servers physisch geschlossen. Eine gemeinsame benutzte Arbeitsdatei, die für Eingabe geöffnet wird, wird physisch geschlossen, wenn die letzte Session sie schließt, d.h. wenn sie eine Dateiende-Bedingung erhält. Beim gleichzeitigen Lesen einer gemeinsam benutzten Arbeitsdatei wird einem READ WORK FILE-Statement nur ein Datei-Datensatz geliefert.

Weitere Informationen

Zu Informationen über Arbeitsdateien (Workfiles) siehe auch *Natural Operations Documentation for Mainframes*.

Arbeitsdatei-Name unter VM/CMS

Unter VM/CMS gilt dasselbe für *operand1* wie unter OS/390 (siehe oben), aber mit folgenden Unterschieden:

- Anstatt dynamischer Zuweisung über MVS SVC 99 wird das CMS-Kommando FILEDEF verwendet, um eine Datei zu definieren.
- HFS-Dateien werden nicht unterstützt.
- JES Spool-Klassen werden nicht unterstützt.
- Außerdem wird die folgende Syntax verwendet:

```
DEFINE WORK FILE n 'fname ftype fmode (options)'
```

Folgendes CMS-Kommando wird generiert:

```
FILEDEF ddname-n DISK fname ftype fmode (options)
```

- Außerdem ist die folgende Syntax zulässig:

```
DEFINE WORK FILE n 'FILEDEF=filedef-parameters'
```

Folgendes CMS-Kommando wird generiert:

```
FILEDEF ddname-n =filedef-parameters
```

Zum Beispiel:

```
DEFINE WORK FILE 5 'FILEDEF=TAP1 SL 2 VOLID BKUP08 (BLKSIZE 20000)'
```

Folgendes CMS-Kommando wird generiert:

```
FILEDEF CMWKF05 TAP1 SL 2 VOLID BKUP08
```

Arbeitsdatei-Name unter BS2000/OSD

Unter BS2000/OSD können Sie für eine Arbeitsdateinummer, die (entweder automatisch in der JCL oder im NETWORK-Makro des Natural-Parametermoduls bzw. mit dem dynamischen Profilparameter WORK) mit Zugriffsmethode AM=STD definiert ist, mit *operand1* einen Dateinamen oder Link-Namen angeben, der dieser Arbeitsdatei zugewiesen werden soll.

In diesem Fall kann *operand1* eine Länge von 1 bis 253 Zeichen haben und eine der folgenden Bedeutungen haben:

- einen BS2000/OSD Link-Namen (1 bis 8 Zeichen)
- einen BS2000/OSD Dateinamen (9 bis 54 Zeichen)
- einen generischen BS2000/OSD Dateinamen (Wildcard)
- einen BS2000/OSD Dateinamen und Link-Namen
- einen generischen BS2000/OSD Dateinamen und Link-Namen (Wildcard)
- *DUMMY.

Es gelten die folgenden Regeln.

- [1] Dateiname und Link-Name können als Positionsparameter oder Schlüsselwort-Parameter angegeben werden. Die entsprechenden Schlüsselwörter sind FILE= und LINK=. Das Vermischen von Positionsparameter und Schlüsselwort-Parameter ist zulässig aber nicht empfehlenswert.
- [2] Eine Zeichenkette mit einer Länge von 1 bis 8 Zeichen ohne Kommas wird als ein Link-Name interpretiert. Diese Notation ist mit früheren Versionen von Natural kompatibel.

Beispiel:

```
DEFINE WORK FILE 1 'W01'
```

Die entsprechende Definition lautet mit Schlüsselwort-Parameter:

```
DEFINE WORK FILE 1 'LINK=W01'
```

- ③ Eine Zeichenkette von 9 bis 54 Zeichen ohne Kommas wird als ein Dateiname interpretiert.

Beispiel:

```
DEFINE WORK FILE 2 'NATURAL31.TEST.WORKFILE02'
```

Die entsprechende Definition lautet mit Schlüsselwort-Parameter:

```
DEFINE WORK FILE 2 'FILE=NATURAL31.TEST.WORKFILE02'
```

- ④ Die folgende Eingabe wird interpretiert, ohne die Länge zu berücksichtigen und bildet deshalb eine Ausnahme von den Regeln 2 und 3:

- Schlüsselwort-Eingabe: LINK=, FILE=
- *DUMMY
- NULLFILE (entspricht *DUMMY)
- *
- *,*

Beispiel: DEFINE WORK FILE 7 'FILE=Y' ist eine gültige Dateizuweisung und kein Link-Name, obwohl die Zeichenkette weniger als 9 Zeichen enthält.

- ⑤ Generische Dateinamen sind wie folgt aufgebaut:

Wnn.userid.tsn.date.time.number

wobei

nn eine Arbeitsdateinummer
userid eine Natural User-ID mit 8 Zeichen
tsn die BS2000/OSD TSN der aktuellen Task mit 4 Ziffern
date gleich *DDMMYYYY*
time gleich *HHIISS*
number eine Zahl mit 5 Ziffern

ist.

- ⑥ Generische Link-Namen sind wie folgt aufgebaut:

NWFnnnnn

nnnnn ist eine 5-stellige Zahl, die nach jeder Generierung eines dynamischen Link-Namen um eins erhöht wird.

- ⑦ Das Ändern der Dateizuweisung für eine Arbeitsdateinummer führt zu einem impliziten Schließen (CLOSE) der bisher zugewiesenen Arbeitsdatei.

Es empfiehlt sich in allen Fällen, außer wenn Sie einen Link-Namen (zum Beispiel: W01) angeben, mit Schlüsselwort-Parametern zu arbeiten. Dies hilft Konflikte bei der Interpretation zusätzlicher Reports zu vermeiden und ist von Bedeutung für Dateinamen mit weniger als 9 Zeichen.

Beispiel:

```
DEFINE WORK FILE 3 'LINK=#W03'
DEFINE WORK FILE 3 'LINK=#W03'
```

Link-Name

Beispiel:

```
DEFINE WORK FILE 1 'LINKW01'
```

bedeutet dasselbe wie

```
DEFINE WORK FILE 1 'LINK=LINKW01'
```

Eine Datei mit dem LINK (Verknüpfung) 'LINKW01' muß zur Laufzeit vorhanden sein. Dieser kann entweder mittels einer JCL vor dem Start von Natural oder durch eine dynamische Zuweisung von der aktuellen Anwendung erstellt werden. Für eine dynamische Zuweisung kann der User-Exit USR2029 in der Library SYSEXT verwendet werden.

Wenn der Link zu einer anderen Datei, z.B. 'W01', vor der Ausführung aktiv war, wird er in Abhängigkeit vom Wert des Profilparameters FREE (mögliche Werte sind ON und OFF) entweder freigegeben oder beibehalten. Die Freigabe erfolgt über einen expliziten RELEASE-Aufruf an den BS2000/OSD-Kommandoprozessor.

Dateiname

Beispiel:

```
DEFINE WORK FILE 2 'NATURAL31.TEST.WORK02'
```

bedeutet dasselbe wie

```
DEFINE WORK FILE 2 'FILE=NATURAL31.TEST.WORK02'
```

Die in *operand1* angegebene Datei wird mittels eines FILE-Makroaufrufs aufgebaut und erbt den Link-Namen, der für die entsprechende Arbeitsdatei vor Ausführung des DEFINE WORK FILE-Statements gültig war.

Generischer Dateiname

Beispiel:

```
DEFINE WORK FILE 21 '*'
```

bedeutet dasselbe wie

```
DEFINE WORK FILE 21 'FILE=*'
```

Eine Datei mit einem nach Regel 4 erstellten Namen wird mittels eines FILE-Makroaufrufs aufgebaut und erbt den Link-Namen, der für die entsprechende Arbeitsdatei vor Ausführung des DEFINE WORK FILE-Statements gültig war.

```
DEFINE WORK FILE 22 'FILE=*,LINK=WFLK22'
```

Eine Datei mit einem nach Regel 4 erstellten Namen wird mittels eines FILE-Makroaufrufs mit dem angegebenen Link-Namen aufgebaut.

Dateiname und Link-Name

Beispiel:

```
DEFINE WORK FILE 11 'NATURAL31.TEST.WORKF11,LNKW11'
```

bedeutet dasselbe wie

```
DEFINE WORK FILE 11 'FILE=NATURAL31.TEST.WORKF11,LINK=LNKW11'
```

was dasselbe bedeutet wie

```
DEFINE WORK FILE 11 'FILE=NATURAL31.TEST.WORKF11,LNKW11'
```

Die in *operand1* angegebene Datei wird mittels eines FILE-Makroaufrufs mit angegebenem Link-Namen aufgebaut und der entsprechenden Arbeitsdateinummer zugewiesen.

Generischer Dateiname und Link-Name

Beispiel:

```
DEFINE WORK FILE 27 '*,*'
```

bedeutet dasselbe wie

```
DEFINE WORK FILE 27 'FILE=*,LINK=*'
```

Eine Datei mit nach Regel 4 und Regel 5 erstelltem Dateinamen und Link-Namen wird mittels eines FILE-Makroaufrufs aufgebaut und der angegebenen Arbeitsdatei 27 zugewiesen.

Anmerkung:

Wenn Dateiname und Link-Name angegeben werden, wird der vorherige Link-Name nicht freigegeben, ungeachtet des Wertes des Profilparameters FREE.

DELETE

```
DELETE [RECORD] [IN] [STATEMENT] [(r)]
```

Verwandte Statements

END TRANSACTION, BACKOUT TRANSACTION, STORE, UPDATE.

Funktion

Das Statement DELETE dient dazu, einen Datensatz von der Datenbank zu löschen.

Hinweise für DL/I-Datenbanken

Mit dem DELETE-Statement kann ein Segment einer DL/I-Datenbank gelöscht werden, wobei gleichzeitig alle “Descendants” dieses Segments gelöscht werden.

Aufgrund von GSAM-Beschränkungen ist das UPDATE-Statement nicht auf GSAM-Datenbanken anwendbar.

Hinweise für SQL-Datenbanken

Mit dem DELETE-Statement können Sie eine Reihe aus einer Datenbank-Tabelle löschen. Das DELETE-Statement entspricht dem SQL-Statement DELETE WHERE CURRENT OF CURSOR-NAME, d.h. nur die zuletzt gelesene Reihe kann gelöscht werden.

Bei den meisten SQL-Datenbanken kann eine mit FIND SORTED BY oder READ LOGICAL gelesene Reihe nicht gelöscht werden.

Hinweise für VSAM-Datenbanken

Das DELETE-Statement ist nicht auf VSAM-ESDS (entry-sequenced datasets) anwendbar.

Referenzieren eines bestimmten Statements (*r*)

Die Notation “(r)” dient dazu, das Statement zu referenzieren, das verwendet wurde, um den zu löschenden Datensatz auszuwählen/zu lesen.

Wenn keine Statement-Referenz angegeben wird, referenziert das DELETE-Statement die jeweils innerste aktive Verarbeitungsschleife, mit der der Datensatz, der gelöscht werden soll, ausgewählt/gelesen wurde.

Anmerkung:

Das DELETE-Statement muß innerhalb der READ- bzw. FIND-Schleife stehen, auf die es sich bezieht.

Einschränkung

Das DELETE-Statement darf nicht mit einem FIND-, READ- oder GET-Statement in derselben Sourcecode-Zeile stehen.

“Hold”-Status

Das Vorhandensein eines DELETE-Statements bewirkt, daß alle Datensätze, die mit dem betreffenden READ- oder FIND-Statement gelesen werden, in den “Hold”-Status gestellt werden.

Die Hold-Logik ist im Kapitel **Datenbankzugriffe** des *Natural Leitfadens zur Programmierung* beschrieben.

Beispiel 1

In diesem Beispiel werden alle Datensätze mit Namen = “ALDEN” gelöscht.

```
/* EXAMPLE 'DELEX1S': DELETE (STRUCTURED MODE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
END-DEFINE
/*****
FIND EMPLOY-VIEW WITH NAME = 'ALDEN'
  DELETE
  END TRANSACTION
/*****
AT END OF DATA
  WRITE NOTITLE *NUMBER 'RECORDS DELETED'
END-ENDDATA
/*****
END-FIND
END
```

Äquivalentes Reporting-Mode-Beispiel: siehe Programm DELEX1R in Library SYSEXRM.

Beispiel 2

Falls in der VEHICLES-Datei für die Person mit Namen ALDEN keine Datensätze gefunden werden, wird von der EMPLOYEES-Datei der Datensatz von ALDEN gelöscht.

```

/* EXAMPLE 'DELEX2S:' DELETE (STRUCTURED MODE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
END-DEFINE
/*****
EMPL.  FIND EMPLOY-VIEW WITH NAME = 'ALDEN'
/*****
VEHC.  FIND VEHIC-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (EMPL.)
      IF NO RECORDS
      DELETE (EMPL.)
      END TRANSACTION
      END-NOREC
      END-FIND
/*****
      END-FIND
/*****
END

```

Äquivalentes Reporting-Mode-Beispiel: siehe Program DELEX2R in Library SYSEXRM.

DISPLAY

DISPLAY [(*rep*)] [*options*] {[/...] [*output-format*] *output-element* }...

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S A G N	A N P I F B D T L G O	ja	nein

Verwandte Statements

WRITE, WRITE TITLE, WRITE TRAILER.

Funktion

Das Statement DISPLAY dient dazu, die Form eines von einem Programm erzeugten Ausgabe-Reports zu bestimmen. Mit dem DISPLAY-Statement werden die Felder angegeben, deren Werte ausgegeben werden sollen. Die Ausgabe erfolgt in Spaltenform, mit einer Spalte pro Feld und einer Spaltenüberschrift.

Report-Spezifikation (*rep*)

Falls nichts anderes angegeben wird, bezieht sich das DISPLAY-Statement auf den ersten ausgegebenen Report (Report 0). Mit der Notation “(*rep*)” können Sie einen bestimmten anderen Report angeben, auf den sich das Statement beziehen soll. Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden.

options

[NOTITLE] [NOHDR] [[AND] [GIVE] [SYSTEM] FUNCTIONS](<i>statement-parameters</i>)

Seitenüberschrift/NOTITLE

Natural generiert für jede über ein DISPLAY-Statement ausgegebene Seite eine Standardkopfzeile. Diese Standardkopfzeile enthält die laufende Seitennummer, Datum und Uhrzeit. Die Uhrzeit wird zu Beginn der Programmausführung (TP-Betrieb) bzw. zu Beginn des Jobs (Batch-Betrieb) gesetzt.

Die Standardkopfzeile kann mit einer eigenen Kopfzeile überschrieben werden, die mit einem WRITE TITLE-Statement angegeben wird. Das Schlüsselwort NOTITLE innerhalb des DISPLAY-Statements bewirkt, daß die Generierung einer Standardkopfzeile unterdrückt wird.

Generierte Standard-Titelzeile wird ausgegeben:

```
DISPLAY NAME
```

Eigene Titelzeile wird ausgegeben:

```
DISPLAY NAME
WRITE TITLE 'USER TITLE'
```

Keine Titelzeile wird ausgegeben:

```
DISPLAY NOTITLE NAME
```

Wenn die NOTITLE-Option verwendet wird, gilt sie für alle DISPLAY-, PRINT- und WRITE-Statements im selben Objekt, die Daten auf denselben Report schreiben.

Spaltenüberschriften/NOHDR

Für jede mittels eines DISPLAY-Statements ausgegebene Spalte von Feldwerten wird eine Spaltenüberschrift ausgegeben; hierbei gilt folgendes:

Sie können mit dem DISPLAY-Statement explizit eine Spaltenüberschrift angeben, und zwar (in Apostrophen) vor dem jeweiligen Feldnamen. Zum Beispiel:

```
DISPLAY 'EMPLOYEE' NAME 'SALARY' SALARY
```

Wenn Sie mit dem DISPLAY-Statement keine Spaltenüberschrift angeben, verwendet Natural die im DEFINE DATA-Statement für das Feld angegebene Spaltenüberschrift. Wenn für ein Datenbankfeld im DEFINE DATA-Statement keine Spaltenüberschrift angegeben ist, wird die im betreffenden DDM definierte Standardüberschrift genommen; ist im DDM keine definiert, wird stattdessen der (im DDM definierte) Feldname als Überschrift verwendet. Wenn für eine Benutzervariable im DEFINE DATA-Statement keine Spaltenüberschrift angegeben ist, wird der Variablenname als Überschrift verwendet. Zur Definition von Spaltenüberschriften vgl. DEFINE DATA-Statement.

```
DISPLAY NAME SALARY #NEW-SALARY
```

Natural unterstreicht die Spaltenüberschriften immer und generiert zwischen Unterstreichung und den ausgegebenen Daten eine Leerzeile.

Enthält ein Programm mehrere DISPLAY-Statements, dann bestimmt das erste DISPLAY-Statement die Spaltenüberschriften; dies wird zur Kompilierungszeit ausgewertet.

Unterdrücken von Spaltenüberschriften

Um die Spaltenüberschrift für ein einzelnes Feld zu unterdrücken, geben Sie vor dem betreffenden Feldnamen die Zeichen '/' (Apostroph-Schrägstrich-Apostroph) an. Beispiel:

```
DISPLAY '/' NAME 'SALARY' SALARY
```

Sollen gar keine Spaltenüberschriften ausgegeben werden, geben Sie das Schlüsselwort NOHDR (für "no header") an:

```
DISPLAY NOHDR NAME SALARY
```

NOHDR gilt nur beim ersten DISPLAY-Statement, da nachfolgende DISPLAY-Statements ohnehin keine Spaltenüberschriften erzeugen können.

Wenn Sie NOTITLE und NOHDR verwenden, müssen Sie sie in der folgenden Reihenfolge angeben:

```
DISPLAY NOTITLE NOHDR NAME SALARY
```

GIVE SYSTEM FUNCTIONS

Die GIVE SYSTEM FUNCTIONS-Klausel dient zur Auswertung der Systemfunktionen von Natural AVER, COUNT, MAX, MIN, NAVER, NCOUNT, NMIN, SUM, TOTAL. Die Systemfunktionen werden ausgewertet, wenn das DISPLAY-Statement, das die GIVE SYSTEM FUNCTIONS-Klausel enthält, ausgeführt wird.

Die Systemfunktionen können anschließend von einem Statement, das aufgrund einer End-of-Page-Bedingung ausgeführt wird, referenziert werden.

Pro Report darf nur *ein* DISPLAY-Statement eine GIVE SYSTEM FUNCTIONS-Klausel enthalten. Die Auswertung von Systemfunktionen über DISPLAY GIVE SYSTEM FUNCTIONS geschieht seitenbezogen, d.h. bei Beginn einer neuen Seite werden alle Systemfunktionen außer TOTAL wieder auf Null gesetzt.

Bei der Verwendung von Systemfunktionen mit einem DISPLAY-Statement, das sich in einer Subroutine befindet, muß die End-of-Page-Verarbeitung innerhalb derselben Subroutine stattfinden.

statement-parameters

Sie können (in Klammern) Session-Parameter setzen, die dann für das DISPLAY-Statement statt der betreffenden mit einem GLOBALS-Kommando, SET GLOBALS- oder FORMAT-Statement gesetzten Parameter gelten.

Werden mehrere Parameter angegeben, müssen sie jeweils durch ein oder mehrere Leerzeichen voneinander getrennt werden. Die Angabe eines Parameters darf sich nicht über zwei Sourcecode-Zeilen erstrecken.

Die hier gültigen Parameter-Einstellungen kommen nur für Variablenfelder in Betracht, haben aber keine Auswirkung auf Text-Konstanten. Möchten Sie Feldattribute für eine Text-Konstante setzen, müssen sie explizit für dieses Element gesetzt werden.

Beispiel

```

DEFINE DATA LOCAL
1  VARI  (A4)   INIT  <'1234'>
END-DEFINE
*
DISPLAY NOHDR          'Text'          '='      VARI          /*      Output
DISPLAY NOHDR (PM=I)  'Text'          '='      VARI          /*      Produced
*
DISPLAY NOHDR          'Text'          (PM=I)   '='      VARI          /*      -----
DISPLAY NOHDR          'Text'          (PM=I)   '='      VARI          /*      Text 1234
DISPLAY NOHDR          'Text'          (PM=I)   '='      VARI          /*      Text 4321
DISPLAY NOHDR          'Text'          (PM=I)   '='      VARI          /*      txeT 4321
END
DISPLAY NOHDR          'Text'          (PM=I)   '='      VARI          /*      txeT 1234

```

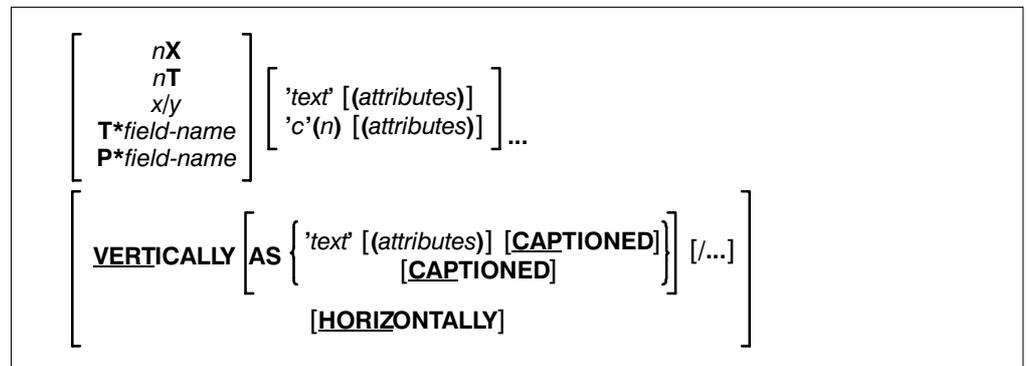
Welche Parameter Sie verwenden dürfen, sowie eine Beschreibung der einzelnen Parameter entnehmen Sie bitte dem Kapitel **Session-Parameter** im *Natural Referenzhandbuch*.

Zeilenvorschub (/)

Ein Schrägstrich “/” innerhalb eines Textelementes bewirkt einen Zeilenvorschub innerhalb des Textes.

Ein Schrägstrich “/” zwischen zwei Ausgabeelementen bewirkt, daß das nachfolgende Element in derselben Spalte ausgegeben wird. Die Überschrift der Spalte wird gebildet, indem die Überschriften der beiden Elemente unmittelbar untereinander ausgegeben werden.

output-format



Feldpositionierung

nX	<p>Mit dieser Notation fügen Sie zwischen den auszugebenden Spalten n Leerstellen ein. n darf nicht "0" sein. Beispiel:</p> <pre>DISPLAY NAME 5X SALARY</pre> <p><i>Anmerkung:</i> <i>Dies gilt nur für Großrechner.</i></p>
nT	<p>Mit dieser Notation setzen Sie Tabulatoren, d.h. die Ausgabe eines Wertes beginnt ab Spalte n. Ein Tabulator, dessen Position bereits durch einen anderen ausgegebenen Wert besetzt ist, darf nicht gesetzt werden. Im folgenden Beispiel wird NAME ab Spalte 25 und SALARY ab Spalte 50 ausgegeben:</p> <pre>DISPLAY 25T NAME 50T SALARY</pre>
x/y	<p>Mit dieser Notation erreichen Sie, daß ein Feld x Zeilen unter der Ausgabe des letzten Statements, und zwar ab Spalte y ausgegeben wird. y darf nicht "0" sein. Zurückpositionieren ist nicht erlaubt.</p>
$T*field-name$	<p>Mit dieser Notation wird die Position eines Feldes nach der Position eines in einem vorhergehenden DISPLAY-Statement ausgegebenen Feldes (<i>field-name</i>) ausgerichtet. Zurückpositionieren ist nicht erlaubt.</p>
$P*field-name$	<p>Mit dieser Notation werden Position <i>und Ausgabezeile</i> eines Feldes nach denen eines in einem vorhergehenden DISPLAY-Statement ausgegebenen Feldes (<i>field-name</i>) ausgerichtet. Dies wird meist bei vertikalen Ausgaben eingesetzt. Zurückpositionieren ist nicht erlaubt.</p>

Spaltenüberschriften

'='	Ein Gleichheitszeichen in Apostrophen unmittelbar vor einem Feld bewirkt, daß entweder die im DDM für das Feld definierte Standardüberschrift bzw. der Feldname (falls im DDM keine Überschrift definiert ist) als Spaltenüberschrift verwendet wird.
'text'	In Apostrophen angegebener <i>text</i> vor einem Feld wird als Spaltenüberschrift verwendet. Beispiel: <pre>DISPLAY 'EMPLOYEE' NAME 'MARITAL/STATUS' MAR-STAT</pre> Werden vor einem Feldnamen mehrere Textelemente angegeben, so wird das letzte als Spaltenüberschrift verwendet und die anderen werden in der Ausgabespalte vor dem Feldwert ausgegeben.
'/'	Ein Schrägstrich in Apostrophen '/' vor einem Feldnamen bewirkt, daß für dieses Feld keine Spaltenüberschrift ausgegeben wird.
'c' (n)	Das Zeichen <i>c</i> wird <i>n</i> -mal unmittelbar vor dem Feldwert ausgegeben. Beispiel: <pre>DISPLAY '**' (5) '=' NAME</pre>

attributes

Dient dazu, den ausgegebenen Feldern/Textelementen Anzeige- und Farbattribute zuzuordnen. *Attributes* können sein:

$\left[\begin{array}{c} \mathbf{B} \\ \mathbf{C} \\ \mathbf{D} \\ \mathbf{I} \\ \mathbf{N} \\ \mathbf{U} \\ \mathbf{V} \end{array} \right] \left[\begin{array}{c} \mathbf{BL} \\ \mathbf{GR} \\ \mathbf{NE} \\ \mathbf{PI} \\ \mathbf{RE} \\ \mathbf{TU} \\ \mathbf{YE} \end{array} \right]$
<div style="display: flex; justify-content: space-around; width: 100%;"> ① ② </div>

① Anzeige-Attribute (siehe Session-Parameter AD im *Natural Referenzhandbuch*).

② Farb-Attribute (siehe Session-Parameter CD im *Natural Referenzhandbuch*).

Vertikale/Horizontale Ausgabe

Mit DISPLAY VERT werden die Werte mehrerer Felder nicht in Spalten nebeneinander sondern in einer Spalte untereinander ausgegeben. Eine neue Spalte wird durch Angabe des Schlüsselwortes VERT oder HORIZ initialisiert.

Die Ausgabe von Spaltenüberschriften wird beim DISPLAY VERT über die AS-Klausel gesteuert:

- Ohne AS-Klausel wird keine Spaltenüberschrift ausgegeben.

```
DISPLAY VERT NAME SALARY
```

- AS *'text'* bewirkt, daß *text* als Spaltenüberschrift ausgegeben wird. Beachten Sie hierbei, daß ein Schrägstrich im *text* einen Zeilenvorschub innerhalb der Überschrift auslöst.

```
DISPLAY VERT AS 'LAST/NAME' NAME
```

- AS *'text'* CAPTIONED bewirkt, daß *text* als Spaltenüberschrift ausgegeben wird und außerdem die Standardspaltenüberschrift bzw. der Feldname in jeder Ausgabezeile dem jeweiligen Feldwert vorangestellt wird.

```
DISPLAY VERT AS 'PERSONS/SELECTED' CAPTIONED NAME FIRST-NAME
```

- AS CAPTIONED bewirkt, daß die Standardspaltenüberschrift (bzw. der Feldname, falls keine Standardüberschrift definiert ist) als Spaltenüberschrift ausgegeben wird.

```
DISPLAY VERT AS CAPTIONED NAME FIRST-NAME
```

Vertikale und horizontale Ausgaben können miteinander kombiniert verwendet werden, wobei der Wechsel von einer Form zur anderen durch die Angabe des jeweiligen Schlüsselwortes (VERT oder HORIZ) erfolgt.

Um die vertikale Ausgabe für ein einzelnes Ausgabeelement auszusetzen, geben Sie vor dem Element einen Gedankenstrich “-” ein. Beispiel:

```
DISPLAY VERT NAME - FIRST-NAME SALARY
```

würde bewirken, daß FIRST-NAME *neben* NAME ausgegeben wird, während SALARY wieder vertikal, d.h. unter NAME, ausgegeben wird.

Normalerweise erzeugt ein DISPLAY-Statement eine horizontale Ausgabe, d.h. die Ausgabe erfolgt in Spalten, die nebeneinander angeordnet sind.

Bei der Generierung der Spaltenüberschriften hat Natural folgende Prioritäten:

- ① Der im DISPLAY-Statement für eine Spaltenüberschrift angegebene *'text'*.
- ② Bei Datenbankfeldern die im DDM definierte Standardspaltenüberschrift, bei Benutzervariablen der Variablenname.
- ③ Bei Datenbankfeldern der Name, unter dem das Feld im DDM definiert ist (wenn für das Datenbankfeld kein Überschriftentext definiert wurde).

Bei Gruppennamen wird eine Gruppen-Spaltenüberschrift für die gesamte Gruppe von Feldern erzeugt. Bei Angabe einer Gruppe kann nur diese Standard-Gruppenüberschrift durch eine eigene überschrieben werden.

Es sind bis zu 15 Spaltenüberschriftenzeilen erlaubt.

Die über ein DISPLAY-Statement erzeugte Ausgabe darf nicht über das Zeilenende hinausgehen; ist dies doch der Fall, gibt Natural eine entsprechende Fehlermeldung aus.

output-element

$\left[\begin{array}{l} \{ 'text' [(attributes)] \} \\ \{ 'c'(n) [(attributes)] \} \dots \\ nX \\ nT \\ x/y \end{array} \right] ['='] \{ operand1 [(parameters)] \}$
--

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S A G N	A N P I F B D T L G O	ja	nein

<i>nX</i>	Ist das gleiche wie unter <i>output-format</i> (siehe oben).
<i>nT</i>	Ist das gleiche wie unter <i>output-format</i> (siehe oben).
<i>x/y</i>	Ist das gleiche wie unter <i>output-format</i> (siehe oben).
<i>'text'</i>	Ist das gleiche wie unter <i>output-format</i> (siehe oben).
<i>'c' (n)</i>	Ist das gleiche wie unter <i>output-format</i> (siehe oben).
<i>'text' '='</i>	Wird <i>'text' '='</i> vor einem Feld angegeben, so wird <i>text</i> unmittelbar vor dem Feldwert ausgegeben. Beispiel: DISPLAY '*****' '=' NAME
<i>attributes</i>	Ist das gleiche wie unter <i>output-format</i> (siehe oben).
<i>operand1</i>	Das auszugebende Feld. <i>Anmerkung für DL/I-Datenbanken:</i> <i>DL/I-AIX-Felder können nur angezeigt werden, wenn ein PCB verwendet wird, in dem das AIX im Parameter PROCSEQ angegeben ist. Ist dies nicht der Fall, gibt Natural zur Laufzeit eine Fehlermeldung aus.</i>
<i>parameters</i>	Unmittelbar nach <i>operand1</i> können Sie in Klammern einen oder mehrere Parameter angeben, die dann für das betreffende Feld statt der entsprechenden mit einem GLOBALS-Kommando, SET GLOBALS- oder FORMAT-Statement gesetzten Parameter gelten. Werden mehrere Parameter angegeben, müssen sie jeweils durch ein oder mehrere Leerzeichen voneinander getrennt werden. Die Angabe eines Parameters darf sich nicht über zwei Sourcecode-Zeilen erstrecken.

Standardwerte

Für ein DISPLAY-Statement gelten folgende Standardwerte:

- ① Die für Ausgaben gültige Standardbreite wird bei der Installation von Natural festgelegt; in der Regel beträgt sie im Batch-Betrieb 132 Stellen und entspricht im TP-Betrieb der Zeilenlänge des Terminals. Sie kann mit dem Session-Parameter LS überschrieben werden. Im TP-Betrieb setzt Natural die Parameter für Zeilenlänge (LS) und Seitenlänge (PS) unter Berücksichtigung der physischen Charakteristika des verwendeten Terminaltyps.
- ② Erfolgt die DISPLAY-Ausgabe auf dem Bildschirm, dann beginnt die Ausgabe in der zweiten physischen Bildschirmspalte (da die erste Spalte für die etwaige Verwendung einer Attributstelle bei einem 3270-er Terminal reserviert werden muß). Wird die DISPLAY-Ausgabe auf Papier ausgedruckt, dann beginnt die Ausgabe ganz links, d.h. in Spalte 1.
- ③ Standardmäßig wird zwischen zwei Ausgabeelementen eine Leerstelle eingefügt. Dieser Standardwert kann mit dem Session-Parameter SF überschrieben werden. Zwischen Ausgabespalten muß mindestens eine Leerspalte (reserviert für Terminal-Attribute) sein.
- ④ Die Breite einer Ausgabespalte richtet sich nach der Länge des Feldes oder der Spaltenüberschrift, je nachdem, was länger ist (es sei denn, der Parameter HW wird verwendet). Ist die Überschrift kürzer als das Feld, wird sie über der Spalte zentriert (es sei denn, mit dem Parameter HC=L bzw. HC=R wird eine linksbündige bzw. rechtsbündige Ausgabe veranlaßt). Ist das Feld kürzer als die Überschrift, wird das Feld linksbündig zur Überschrift ausgerichtet. Bei alphanumerischen Feldern werden die Feldwerte linksbündig im Feld ausgegeben, bei numerischen rechtsbündig. Mit dem Parameter AD=L kann auch bei numerischen Feldern eine linksbündige Ausgabe erreicht werden, bzw. mit AD=R bei alphanumerischen Feldern eine rechtsbündige Ausgabe. Bei vertikalen Ausgaben richtet sich die Breite einer Spalte nach dem längsten Feldwert bzw. der längsten Überschrift (es sei denn, der Parameter HW wird verwendet).
- ⑤ Bei der Ausgabe eines numerischen Feldes wird eine Stelle vor dem Feld für die Ausgabe eines Vorzeichens reserviert. Die Vorzeichenstelle kann mit dem Session-Parameter SG unterdrückt werden.
- ⑥ Natural prüft *vor* der Ausführung eines DISPLAY-Statements, wann ein Seitenumbruch erforderlich ist. *Während* der Ausführung des DISPLAY-Statements werden keine Kopf- oder Fußzeilen generiert.

Beispiel 1

```
/* EXAMPLE 'DISEX1:' DISPLAY (USING NX, NT NOTATION)
/*****
LIMIT 4
READ EMPLOYEES BY NAME
  DISPLAY NOTITLE 5X NAME 50T JOB-TITLE
/*****
END
```

NAME	CURRENT POSITION
ABELLAN	MAQUINISTA
ACHIESON	DATA BASE ADMINISTRATOR
ADAM	CHEF DE SERVICE
ADKINSON	SALES PERSON

Beispiel 2

```

/* EXAMPLE 'DISEX2' DISPLAY (GIVE SYSTEM FUNCTIONS)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
  2 SALARY (1)
  2 CURR-CODE (1)
END-DEFINE
/*****
LIMIT 15
FORMAT PS=15
READ EMPLOY-VIEW
  DISPLAY GIVE SYSTEM FUNCTIONS
    PERSONNEL-ID NAME FIRST-NAME SALARY (1) CURR-CODE (1)
  AT END OF PAGE
    WRITE / 'SALARY STATISTICS:'
      / 7X 'MAXIMUM:' MAX(SALARY(1)) CURR-CODE (1)
      / 7X 'MINIMUM:' MIN(SALARY(1)) CURR-CODE (1)
      / 7X 'AVERAGE:' AVER(SALARY(1)) CURR-CODE (1)
    END-ENDPAGE
  END-READ
/*****
END

```

PERSONNEL ID	NAME	FIRST-NAME	ANNUAL SALARY	CURRENCY CODE
50005600	MORENO	HUMBERTO	165810	FRA
50005500	BLOND	ALEXANDRE	172000	FRA
50005300	MAIZIERE	ELISABETH	166900	FRA
50004900	CAUDAL	ALBERT	167350	FRA
50004600	VERDIE	BERNARD	170100	FRA
50004300	GUERIN	MICHELE	163900	FRA
50004200	VAUZELLE	BERNARD	159790	FRA
50004100	CHAPUIS	ROBERT	169900	FRA
50004000	MONTASSIER	JEAN	175550	FRA
SALARY STATISTICS:				
	MAXIMUM:	175550	FRA	
	MINIMUM:	159790	FRA	
	AVERAGE:	167922	FRA	

Beispiel 3

```

* EXAMPLE 'DISEX3': DISPLAY (USING P* NOTATION)
*****
DEFINE DATA LOCAL
  1 EMP-VIEW VIEW OF EMPLOYEES
    2 NAME
    2 SALARY (1)
    2 BIRTH
    2 CITY
END-DEFINE
*
LIMIT 2
READ EMP-VIEW BY CITY FROM 'N'
  DISPLAY NOTITLE NAME CITY
    VERT AS 'BIRTH/SALARY' BIRTH (EM=YYYY-MM-DD) SALARY (1)
SKIP 1
AT BREAK OF CITY
  DISPLAY P*SALARY (1) AVER(SALARY (1))
  SKIP 1
END-BREAK
END-READ
END

```

NAME	CITY	BIRTH SALARY
WILCOX	NASHVILLE	1970-01-01 38000
MORRISON	NASHVILLE	1949-07-10 36000
		37000

Beispiel 4

```

/* EXAMPLE 'DISEX4:' DISPLAY (USING 'TEXT', 'C(N)' NOTATION, AND
/*                               ATTRIBUTE NOTATION)
/*****
LIMIT 4
READ EMPLOYEES BY DEPT FROM 'T'
  IF LEAVE-DUE GT 40
    DISPLAY NOTITLE 'EMPLOYEE' NAME          /* OVERRIDE STANDARD HEADER
      'LEAVE ACCUMULATED' LEAVE-DUE /* OVERRIDE STANDARD HEADER
      '*' (10) (I)                       /* DISPLAY 10 '*' INTENSIFIED
  ELSE
    DISPLAY NAME LEAVE-DUE
/*****
END

```

EMPLOYEE	LEAVE ACCUMULATED	
LAVENDA	33	
BOYER	33	
CORREARD	45	*****
BOUVIER	19	

Beispiel 5

```

/* EXAMPLE 'DISEX5': DISPLAY (HORIZONTAL DISPLAY)
/*****
LIMIT 4
READ EMPLOYEES BY NAME
  DISPLAY NOTITLE NAME JOB-TITLE SALARY (1:2) 'CURR-CODE (1:2)
  SKIP 1
/*****
END

```

NAME	CURRENT POSITION	ANNUAL SALARY	CURRENCY CODE
ABELLAN	MAQUINISTA	1450000	PTA
		1392000	PTA
ACHIESON	DATA BASE ADMINISTRATOR	10500	UKL
		11300	UKL
ADAM	CHEF DE SERVICE	159980	FRA
		0	
ADKINSON	SALES PERSON	36000	USD
		33100	USD

Beispiel 6

```

/* EXAMPLE 'DISEX6': DISPLAY (VERTICAL AND HORIZONTAL DISPLAY)
/*****
LIMIT 1
READ EMPLOYEES BY NAME
  DISPLAY NOTITLE VERT AS CAPTIONED
    NAME CITY 'POSITION' JOB-TITLE
    HORIZ 'SALARY' SALARY (1:2) 'CURRENCY' CURR-CODE (1:2)
/*****
SKIP 1
END

```

NAME CITY POSITION	SALARY	CURRENCY
ABELLAN	1450000	PTA
MADRID	1392000	PTA
MAQUINISTA		

Beispiel 7

```

/* EXAMPLE 'DISEX7': DISPLAY (USING STATEMENT/ELEMENT PARAMETERS)
/*****
LIMIT 3
READ EMPLOYEES BY NAME
  DISPLAY NOTITLE (AL=16 GC=+ NL=8 SF=3 UC==)
    PERSONNEL-ID NAME TELEPHONE (LC=< TC=>)
/*****
END

```

PERSONNEL ID	NAME	+++++TELEPHONE+++++	
		AREA CODE	TELEPHONE
=====	=====	=====	=====
60008339	ABELLAN	<1 >	<4356726 >
30000231	ACHIESON	<0332 >	<523341 >
50005800	ADAM	<1033 >	<44864858 >

DIVIDE

DIVIDE [ROUNDED] operand1 INTO operand2 [GIVING operand3]

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S A N	N P I F	ja	nein
Operand2	C S A NM*	N P I F	ja	nein
Operand3	S A	A N P I F B	ja	ja

* "N" wenn GIVING-Klausel verwendet wird, "M" wenn GIVING-Klausel nicht verwendet wird.

DIVIDE operand1 INTO operand2 [GIVING operand3] REMAINDER operand4

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S A N	N P I	ja	nein
Operand2	C S A N	N P I	ja	nein
Operand3	S A	A N P I F B	ja	ja
Operand4	S A	A N P I F B	ja	ja

Verwandtes Statement

COMPUTE.

Funktion

Mit dem Statement DIVIDE können Sie einen Operanden durch einen anderen dividieren.

Ergebnisfeld

Das Ergebnisfeld kann entweder eine Benutzervariable oder ein Datenbankfeld sein.

Wenn Sie das Schlüsselwort `ROUNDED` angeben, erhalten Sie das Ergebnis gerundet.

Verwenden Sie eine `GIVING`-Klausel, wird das Ergebnis in *operand3* ausgegeben; *operand2* bleibt unverändert. Ohne `GIVING`-Klausel wird das Ergebnis in *operand2* ausgegeben. Wenn *operand2* eine Konstante oder eine nicht-modifizierbare Natural-Systemvariable ist, muß eine `GIVING`-Klausel mit *operand3* als Ergebnisfeld angegeben werden.

Wird ein Datenbankfeld als Ergebnisfeld verwendet, ändert sich durch die Division lediglich der programmintern verwendete Wert des Feldes; der in der Datenbank gespeicherte Wert des Feldes wird davon nicht berührt.

Die Anzahl der Dezimalstellen des Ergebnisses richtet sich nach dem Dividenten (also *operand2*, wenn keine `GIVING`-Klausel verwendet wird, oder *operand3*, wenn eine `GIVING`-Klausel verwendet wird).

Informationen zur Genauigkeit der Ergebnisse siehe Abschnitt **Genauigkeit von Ergebnissen bei arithmetischen Operationen** im *Natural Referenzhandbuch*.

Division durch Null

Wird eine Teilung durch Null versucht, d.h. wenn der Divisor (*operand1*), also die Zahl durch die geteilt wird, "0" ist, wird entweder eine entsprechende Fehlermeldung oder als Ergebnis "0" ausgegeben, je nachdem wie der Session-Parameter `ZD` (der im *Natural Referenzhandbuch* beschrieben ist) gesetzt ist.

REMAINDER-Option

Das Schlüsselwort REMAINDER bewirkt, daß der nach einer (ungerundeten) Division verbleibende Rest in *operand4* ausgegeben wird.

Wenn GIVING *und* REMAINDER verwendet werden, darf keiner der vier Operanden ein Array-Bereich sein.

Intern wird der Rest wie folgt berechnet:

- ① Der Quotient der Division von *operand1* und *operand2* wird berechnet.
- ② Der Quotient wird mit *operand1* multipliziert.
- ③ Das Produkt dieser Multiplikation wird von *operand2* subtrahiert.
- ④ Das Ergebnis dieser Subtraktion wird *operand4* zugewiesen.

Für jeden dieser Schritte gelten die unter **Genauigkeit von Ergebnissen bei arithmetischen Operationen** im *Natural-Referenzhandbuch* beschriebenen Regeln.

Beispiel

```

/* EXAMPLE 'DIVEX1': DIVIDE
/*****
DEFINE DATA LOCAL
1 #A (N7) INIT <20>
1 #B (N7)
1 #C (N3.2)
1 #D (N1)
1 #E (N1) INIT <3>
1 #F (N1)
END-DEFINE
/*****
DIVIDE 5 INTO #A
WRITE NOTITLE 'DIVIDE 5 INTO #A' 20X '=' #A
/*****
RESET INITIAL #A
DIVIDE 5 INTO #A GIVING #B
WRITE 'DIVIDE 5 INTO #A GIVING #B' 10X '=' #B
/*****
DIVIDE 3 INTO 3.10 GIVING #C
WRITE 'DIVIDE 3 INTO 3.10 GIVING #C' 8X '=' #C
/*****
DIVIDE 3 INTO 3.1 GIVING #D
WRITE 'DIVIDE 3 INTO 3.1 GIVING #D' 9X '=' #D
/*****
DIVIDE 2 INTO #E REMAINDER #F
WRITE 'DIVIDE 2 INTO #E REMAINDER #F' 7X '=' #E '=' #F
/*****
END

```

DIVIDE 5 INTO #A	#A:	4
DIVIDE 5 INTO #A GIVING #B	#B:	4
DIVIDE 3 INTO 3.10 GIVING #C	#C:	1.03
DIVIDE 3 INTO 3.1 GIVING #D	#D:	1
DIVIDE 2 INTO #E REMAINDER #F	#E:	1
	#F:	1

DO/DOEND

Anmerkung:

Die Statements DO und DOEND dürfen nur im Reporting Mode verwendet werden.

DO statement... DOEND

Funktion

Mit den folgenden Statements geben Sie eine logische Bedingung an:

- AT BREAK
- AT END OF DATA
- AT END OF PAGE
- AT START OF DATA
- AT TOP OF PAGE
- BEFORE BREAK PROCESSING
- FIND ... IF NO RECORDS FOUND
- IF
- IF SELECTION
- ON ERROR
- READ WORK FILE ... AT END OF FILE

Werden in Abhängigkeit von einer solchen logischen Bedingung mehrere Statements ausgeführt, so müssen im *Reporting Mode* diese Statements in die Statements DO und DOEND eingeschlossen werden.

Einschränkungen

Die Statements WRITE TITLE, WRITE TRAILER und Statements, die mit AT beginnen, dürfen *innerhalb* einer DO/DOEND-Konstruktion nicht verwendet werden.

Wenn Sie innerhalb einer DO/DOEND-Konstruktion eine Verarbeitungsschleife initiieren, müssen Sie sie vor dem DOEND-Statement wieder schließen.

Beispiel

Siehe Programm DOEEX1 in Library SYSEXRM.

DOWNLOAD

Dieses Statement ist nur in Zusammenhang mit Natural Connection verfügbar. Es ist in der Natural Connection-Dokumentation beschrieben.

EJECT

Syntax 1

$$\text{EJECT } \left\{ \begin{array}{l} \text{ON} \\ \text{OFF} \end{array} \right\} [(rep)]$$

Funktion

EJECT ON/OFF mit Report-Spezifikation (Online- und Batch-Verarbeitung)

“EJECT OFF (*rep*)” bewirkt, daß für den angegebenen Report *kein* Seitenvorschub (außer einem mit Syntax 2 des EJECT-Statements angegebenen) ausgeführt wird.

“EJECT ON (*rep*)” bewirkt, daß Seitenvorschübe für den angegebenen Report ausgeführt werden.

EJECT ON/OFF ohne Report-Spezifikation (nur Batch-Verarbeitung)

EJECT ON/OFF — ohne (*rep*)-Notation — kann im Batch-Betrieb dazu verwendet werden, den Seitenvorschub zwischen den bei der Ausführung eines Programms erzeugten Ausgabelisten zu steuern.

EJECT ON (gilt auch standardmäßig) bewirkt, daß Natural jeweils zwischen der Sourceprogramm-Auflistung, dem Ausgabe-Report und der Meldung “EXECUTION COMPLETED” einen Seitenvorschub ausführt.

EJECT OFF bewirkt, daß keiner der oben genannten Seitenvorschübe ausgeführt wird. EJECT OFF gilt solange, bis es durch ein nachfolgendes EJECT ON-Statement wieder zurückgenommen wird.

Report-Spezifikation (*rep*)

Mit der Notation “(*rep*)” kann ein bestimmter anderer Report angegeben werden, auf den sich das Statement beziehen soll. Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden. Falls nichts anderes angegeben wird, bezieht sich das EJECT-Statement auf den ersten Report (Report 0).

Syntax 2

$\text{EJECT } [(rep)] \left[\left[\begin{array}{c} \text{IF} \\ \text{WHEN} \end{array} \right] \text{ LESS } [\text{THAN}] \textit{operand1} [\text{LINES}] [\text{LEFT}] \right]$
--

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	N P I	ja	nein

Funktion

Diese Form des EJECT-Statements kann dazu verwendet werden, einen Seitenvorschub auszulösen, ohne daß eine “End-of-Page”- oder “Top-of-Page”-Verarbeitung durchgeführt oder auf der neuen Seite eine Titel- oder Kopfzeile generiert wird.

Report-Spezifikation (*rep*)

Mit der Notation “(*rep*)” kann ein bestimmter anderer Report angegeben werden, auf den sich das Statement beziehen soll. Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden. Falls nichts anderes angegeben wird, bezieht sich das EJECT-Statement auf den ersten ausgegebenen Report (Report 0).

IF LESS THAN *operand1* LINES LEFT

Ein Seitenvorschub wird nur ausgeführt, wenn die aktuelle Zeile für die Seite größer als die Seitengröße minus *operand1* ist. *Operand1* kann als numerische Konstante oder als Variable angegeben werden.

Verarbeitung

Die Ausführung eines EJECT-Statements löst keine Ausführung der mit AT TOP OF PAGE, AT END OF PAGE, WRITE TITLE oder WRITE TRAILER verknüpften Statements aus. Ebenso wenig beeinflusst es die Auswertung von Systemfunktionen in einem DISPLAY-Statement mit GIVE SYSTEM FUNCTIONS-Klausel.

Das Statement EJECT bewirkt lediglich, daß eine neue physische Ausgabeseite begonnen wird. Es bewirkt außerdem, daß der Wert der Natural-Systemvariablen *LINE-COUNT wieder auf "1" gesetzt wird, hat aber keinen Einfluß auf die Natural-Systemvariable *PAGE-NUMBER.

Beispiel

```

/* EXAMPLE 'EJTEX1:' EJECT
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 JOB-TITLE
END-DEFINE
/*****
FORMAT PS=15
LIMIT 9
READ EMPLOY-VIEW BY CITY
/*****
AT START OF DATA
  EJECT
  WRITE /// 20T '%' (29) /
           20T '%%'                               47T '%%' /
           20T '%%' 3X 'REPORT OF EMPLOYEES' 47T '%%' /
           20T '%%' 3X ' SORTED BY CITY   ' 47T '%%' /
           20T '%%'                               47T '%%' /
           20T '%' (29) /
  EJECT
END-START
EJECT WHEN LESS THAN 3 LINES LEFT
/*****
WRITE '*' (64)
DISPLAY NOTITLE NOHDR CITY NAME JOB-TITLE 5X *LINE-COUNT
WRITE '*' (64)
END-READ
END

```

```

%% %% %% %% %% %% %% %% %% %% %% %% %% %% %% %% %% %% %% %%
%%                               %%
%%  REPORT OF EMPLOYEES        %%
%%    SORTED BY CITY          %%
%%                               %%
%% %% %% %% %% %% %% %% %% %% %% %% %% %% %% %% %% %% %% %%
  
```

```

*****
AIKEN                SENKO                PROGRAMMER                2
*****
AIX EN OTHE.        GODEFROY                COMPTABLE                5
*****
AJACCIO             CANALE                CONSULTANT                8
*****
ALBERTSLUND        PLOUG                KONTORASSISTENT        11
*****
ALBUQUERQUE        HAMMOND                SECRETARY                14
*****
  
```

```

*****
ALBUQUERQUE        ROLLING                MANAGER                2
*****
ALBUQUERQUE        FREEMAN                MANAGER                5
*****
ALBUQUERQUE        LINCOLN                ANALYST                8
*****
ALFRETON           GOLDBERG                JUNIOR                11
*****
  
```

END

```
{ END }  
.
```

Funktion

Das Statement END dient dazu, das physische Ende eines Natural-Programms zu kennzeichnen.

Im *Reporting Mode* werden durch das END-Statement alle noch aktiven Verarbeitungsschleifen (die noch nicht durch ein LOOP-Statement beendet wurden) geschlossen.

period – .

Falls Sie statt END einen Punkt (.) verwenden und sich in derselben Zeile noch andere Statements befinden, müssen Sie dem Punkt mindestens ein Leerzeichen voranstellen.

Hinweise zur Programmausführung

Wird ein END-Statement in einem Hauptprogramm (einem Programm, das auf Stufe 1 ausgeführt wird) ausgeführt, so wird eine abschließende “End-of-Page”-Verarbeitung ausgeführt sowie für alle vom Benutzer ausgelösten Gruppenwechsel (PERFORM BREAK PROCESSING), die sich nicht durch Referenzierung (Statement-Label oder Sourcecode-Zeilenummer) auf eine bestimmte Verarbeitungsschleife beziehen, eine abschließende Gruppenwechsel-Verarbeitung.

Die Ausführung eines END-Statements in einem Subprogramm oder einem mit FETCH RETURN aufgerufenen Programm bewirkt lediglich, daß die Kontrolle wieder an das aufrufende Programm übergeben wird.

Beispiele

Siehe jedes beliebige Programm in diesem Kapitel.

END TRANSACTION

END [OF] TRANSACTION [*operand1...*]

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S N	A N P I F B D T	ja	nein

Verwandte Statements

GET TRANSACTION DATA, BACKOUT TRANSACTION, STORE, UPDATE, DELETE.

Funktion

Das Statement END TRANSACTION dient dazu, das Ende einer logischen Transaktion zu markieren. Eine logische Transaktion ist die kleinste (vom Benutzer definierte) logische Arbeitseinheit, die vollständig ausgeführt werden muß, damit die logische Konsistenz der Daten auf der Datenbank gewährleistet ist.

Die erfolgreiche Ausführung eines END TRANSACTION-Statements bewirkt, daß alle im Verlaufe der Transaktion durchgeführten Datenänderungen physisch auf der Datenbank durchgeführt worden sind (bzw. werden) und von einem anschließenden Abbruch, sei er durch den Benutzer, Natural, die Datenbank oder das Betriebssystem herbeigeführt, nicht mehr beeinflußt werden können.

Wenn das END TRANSACTION-Statement nicht erfolgreich ausgeführt wird, d.h. wenn die logische Transaktion nicht vollständig ausgeführt ist, werden alle im Laufe der Transaktion bereits durchgeführten Datenänderungen automatisch wieder rückgängig gemacht.

END TRANSACTION bewirkt außerdem, daß alle während der Transaktion im "Hold"-Status gehaltenen Datensätze wieder freigegeben werden.

Die Ausführung des END TRANSACTION-Statements kann an eine logische Bedingung geknüpft werden.

Weitere Informationen hierzu finden Sie im Kapitel **Datenbankzugriffe** des *Natural Leitfadens zur Programmierung*.

Betroffene Datenbanken

Ein END TRANSACTION-Statement *ohne* Transaktionsdaten (d.h. ohne *operand1*) wird nur ausgeführt, wenn eine Datenbanktransaktion unter Kontrolle von Natural stattgefunden hat. Für welche Datenbanken das Statement ausgeführt wird, hängt davon ab, wie der Natural-Profilparameter ET (siehe *Natural Referenzhandbuch* oder *Natural Operations Documentation*) gesetzt ist.

Bei ET=OFF wird das Statement nur für die von der Transaktion betroffene Datenbank ausgeführt; bei ET=ON wird es für alle Datenbanken ausgeführt, die seit der letzten Ausführung eines BACKOUT TRANSACTION- oder END TRANSACTION-Statements referenziert wurden.

Ein END TRANSACTION-Statement *mit* Transaktionsdaten (d.h. mit Angabe von *operand1*) wird immer ausgeführt, und die Transaktionsdaten werden wie unten beschrieben auf einer bestimmten Datenbank gespeichert. Für welche Datenbanken das Statement außerdem ausgeführt wird, hängt vom ET-Parameter (siehe oben) ab.

Speicherung von Transaktionsdaten (*operand1*)

Bei einer Transaktion auf einer Adabas-Datenbank, oder auf einer DL/I-Datenbank in einer batch-orientierten BMP-Region (nur in IMS-Umgebungen) können Sie mit diesem Statement auch transaktionsbezogene Daten speichern. Diese Daten dürfen maximal 2000 Bytes lang sein und können mit einem GET TRANSACTION DATA-Statement wieder gelesen werden.

Die Transaktionsdaten werden auf die mit dem Profilparameter ETDB (siehe *Natural Referenzhandbuch* oder *Natural Operations Documentation*) angegebene Datenbank geschrieben.

Ist der ETDB-Parameter nicht gesetzt, werden die Transaktionsdaten auf die mit dem Profilparameter UDB angegebene Datenbank geschrieben — außer auf Großrechnern: hier werden sie auf die Datenbank geschrieben, auf der sich die Natural-Security-Systemdatei (FSEC) befindet (ist FSEC nicht angegeben, dann ist sie identisch mit der Natural-Systemdatei FNAT; ist Natural Security nicht installiert, dann werden die Transaktionsdaten auf die Datenbank geschrieben, auf der sich FNAT befindet).

Hinweise für DL/I-Datenbanken

Da die PSB-Initialisierung durch eine “Syncpoint”-Anfrage beendet wird, speichert Natural die PSB-Position, bevor das END TRANSACTION-Statement ausgeführt wird. Bevor das nächste Kommando ausgeführt wird, re-initialisiert Natural den PSB und versucht, die PCB-Position so zu setzen, wie sie vor dem END TRANSACTION-Statement war. Die PCB-Position kann vorverlegt werden, falls in der Zeit zwischen END TRANSACTION und dem nachfolgenden Kommando ein adressiertes Segment gelöscht wurde.

Hinweise für SQL-Datenbanken

Da die meisten SQL-Datenbanken bei Beendigung einer logischen Arbeitseinheit alle Cursor schließen, darf ein END TRANSACTION-Statement nicht innerhalb einer datenbankverändernden Verarbeitungsschleife stehen, sondern muß nach einer solchen plaziert werden.

Hinweise für VSAM-Datenbanken

Informationen zur Transaktionslogik, die beim Zugriff auf VSAM gilt, finden Sie in der *Natural for VSAM Documentation*.

Einschränkung

Mit Entire System Server kann dieses Statement nicht verwendet werden.

Beispiel 1

```

/* EXAMPLE 'ETREX1S:' END TRANSACTION (STRUCTURED MODE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 COUNTRY
END-DEFINE
/*****
FIND EMPLOY-VIEW WITH CITY = 'BOSTON'
ASSIGN COUNTRY = 'USA'
UPDATE
END TRANSACTION
/*****
AT END OF DATA
WRITE NOTITLE *NUMBER 'RECORDS UPDATED'
END-ENDDATA
/*****
END-FIND
END

```

7 RECORDS UPDATED

Äquivalentes Reporting-Mode-Beispiel: siehe Programm ETREX1R in Library SYSEXRM.

Beispiel 2

```

/* EXAMPLE 'ETREX2:' END TRANSACTION (WITH ET DATA)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
  2 CITY
1 #PERS-NR (A8) INIT <' '>
END-DEFINE
/*****
REPEAT
  INPUT 'ENTER PERSONNEL NUMBER TO BE UPDATED:' #PERS-NR
  IF #PERS-NR = ' '
    ESCAPE BOTTOM
  END-IF
/*****
FIND EMPLOY-VIEW PERSONNEL-ID = #PERS-NR
  INPUT (AD=M)  NAME / FIRST-NAME / CITY
  UPDATE
  END TRANSACTION #PERS-NR
END-FIND
/*****
END-REPEAT
END

```

ENTER PERSONNEL NUMBER TO BE UPDATED: 20027800

NAME LAWLER
FIRST-NAME SUNNY
CITY MILWAUKEE

ESCAPE

Structured-Mode-Syntax

$$\text{ESCAPE} \left\{ \begin{array}{l} \text{TOP} \\ \text{BOTTOM } [(r)] \text{ [IMMEDIATE]} \\ \text{ROUTINE [IMMEDIATE]} \end{array} \right\}$$

Reporting-Mode-Syntax

$$\text{ESCAPE} \left[\begin{array}{l} \text{TOP} \\ \underline{\text{BOTTOM}} [(r)] \text{ [IMMEDIATE]} \\ \text{ROUTINE [IMMEDIATE]} \end{array} \right]$$

Funktion

Das Statement ESCAPE dient dazu, den linearen Ablauf der Ausführung einer Verarbeitungsschleife oder eines Unterprogramms zu unterbrechen.

Mit den Schlüsselwörtern TOP, BOTTOM und ROUTINE geben Sie an, wo die Verarbeitung nach dem ESCAPE-Statement fortgesetzt werden soll.

Ein ESCAPE TOP- bzw. ESCAPE BOTTOM-Statement bezieht sich immer auf die innerste gerade aktive Verarbeitungsschleife. Das ESCAPE-Statement muß nicht unbedingt innerhalb der Schleife stehen.

Befindet sich das ESCAPE TOP/BOTTOM-Statement in einem Unterprogramm (Subroutine, Subprogramm oder mit FETCH RETURN aufgerufenes Programm), werden die innerhalb der Verarbeitungsschleife aufgerufenen Unterprogramme automatisch beendet.

ESCAPE TOP

TOP bedeutet, daß die Verarbeitung am Anfang der Verarbeitungsschleife fortgesetzt werden soll, d.h. die Schleife wird erneut von Anfang an durchlaufen.

ESCAPE BOTTOM

BOTTOM bedeutet, daß die Verarbeitung mit dem ersten auf das Ende der Schleife folgenden Statement fortgesetzt wird. Die Schleife wird beendet und etwaige schleifenabschließende Verarbeitungen (BREAK und END OF DATA) durchgeführt, und zwar für alle betroffenen, d.h. innerhalb der Schleife befindlichen und folglich ebenfalls beendeten Verarbeitungsschleifen.

Wird mit BOTTOM ein Statement-Label bzw. eine Sourcecode-Zeilenummer (*r*) angegeben, so wird die Verarbeitung mit dem ersten Statement, das auf die so referenzierte Verarbeitungsschleife folgt, fortgesetzt.

Wenn Sie das Schlüsselwort IMMEDIATE angeben, werden keine abschließenden schleifenbeendenden Verarbeitungen durchgeführt.

Im *Reporting Mode* gilt standardmäßig ESCAPE BOTTOM, wenn Sie nichts anderes angeben.

ESCAPE ROUTINE

Diese Option bewirkt, daß das aktive Natural-Unterprogramm, das entweder über PERFORM, CALLNAT, FETCH RETURN oder als Hauptprogramm aufgerufen wurde, die Kontrolle abgibt.

Im Falle einer Subroutine wird die Verarbeitung mit dem ersten Statement fortgesetzt, das auf das Statement folgt, mit dem die Subroutine aufgerufen wurde. Im Falle eines Hauptprogramms gelangt Natural in den Kommando-Modus.

Alle aktiven Schleifen innerhalb des Unterprogramms werden beendet und schleifenabschließende Verarbeitungen (BREAK und END OF DATA) sowie vom Benutzer bestimmte Gruppenwechsel-Verarbeitungen (PERFORM BREAK) durchgeführt, und zwar für alle betroffenen Verarbeitungsschleifen. Steht das ESCAPE ROUTINE-Statement in einem auf Stufe 1 ausgeführten Hauptprogramm, wird außerdem eine abschließende "End-of-Page"-Verarbeitung durchgeführt.

Wenn Sie das Schlüsselwort IMMEDIATE angeben, werden keine abschließenden schleifenbeendenden Verarbeitungen durchgeführt.

Weitere Hinweise

In einer Verarbeitungsschleife können mehrere ESCAPE-Statements enthalten sein.

Die Ausführung eines ESCAPE-Statements kann an eine logische Bedingung geknüpft werden.

Befindet sich das ESCAPE-Statement in einem AT END OF DATA-, AT BREAK- oder AT END OF PAGE-Block, so wird die Verarbeitung des betreffenden Blocks abgebrochen und die ESCAPE-Verarbeitung wie angegeben fortgesetzt.

Wird das ESCAPE-Statement während einer IF NO RECORDS FOUND-Bedingung ausgeführt, werden keine schleifenabschließenden Verarbeitungen durchgeführt (entspricht ESCAPE IMMEDIATE).

Beispiel

```

/* EXAMPLE 'ESCEX1S': ESCAPE (STRUCTURED MODE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 FIRST-NAME
  2 NAME
  2 AREA-CODE
  2 PHONE
1 #CITY (A20) INIT <' '>
1 #CNTL (A1) INIT <' '>
END-DEFINE
/*****
RPT. REPEAT
  INPUT 'ENTER VALUE FOR CITY: ' #CITY
    / '(OR '.' TO TERMINATE)'
  IF #CITY = '.'
    STOP
  END-IF
/*****
FND. FIND EMPLOY-VIEW WITH CITY = #CITY
/*****
  IF NO RECORDS FOUND
    WRITE 'NO RECORDS FOUND'
    ESCAPE BOTTOM (FND.)
  END-NOREC
  AT START OF DATA
    INPUT (AD=0) 'RECORDS FOUND:' *NUMBER //
      'ENTER 'D' TO DISPLAY RECORDS' #CNTL (AD=M)
    IF #CNTL NE 'D'
      ESCAPE BOTTOM (FND.)
    END-IF
  END-START
/*****
  DISPLAY NOTITLE NAME FIRST-NAME PHONE
  END-FIND
  END-REPEAT
/*****
END

```

```
ENTER VALUE FOR CITY:  PARIS
(OR '.' TO TERMINATE)
```

```
RECORDS FOUND:      24
```

```
ENTER 'D' TO DISPLAY RECORDS  D
```

NAME	FIRST-NAME	TELEPHONE	
		AREA CODE	TELEPHONE
MAIZIERE	ELISABETH	1033	46758304
MARX	JEAN-MARIE	1033	40738871
REIGNARD	JACQUELINE	1033	48472153
RENAUD	MICHEL	1033	46055008
REMOUE	GERMAINE	1033	36929371
LAVENDA	SALOMON	1033	40155905
BROUSSE	GUY	1033	37502323
GIORDA	LOUIS	1033	37497316
SIECA	FRANCOIS	1033	40487413
CENSIER	BERNARD	1033	38070268
DUC	JEAN-PAUL	1033	38065261
CAHN	RAYMOND	1033	43723961
MAZUY	ROBERT	1033	44286899
VALLY	ALAIN	1033	47326249
BRETON	JEAN-MARIE	1033	48467146
GIGLEUX	JACQUES	1033	40477399
XOLIN	CHRISTIAN	1033	46060015

Äquivalentes Reporting-Mode-Beispiel: siehe Programm ESCEX1R in Library SYSEXRM.

EXAMINE

```

EXAMINE [FULL [VALUE [OF]]] { SUBSTRING (operand1operand1,operand2,operand3) }
[FOR][FULL [VALUE [OF]]][PATTERN] operand4
[DELIMITERS-option]
{[DELETE/REPLACE-clause][GIVING-clause...]}

```

Sie können die DELETE/REPLACE-Klausel oder die GIVING-Klausel oder beide verwenden, müssen aber wenigstens eine der beiden verwenden.

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C* S A	A	ja	nein
Operand2	C S	N P I	ja	nein
Operand3	C S	N P I	ja	nein
Operand4	C S	A	ja	nein

* *Operand1* darf nur eine Konstante sein, wenn Sie die GIVING-Klausel verwenden, aber nicht, wenn Sie die DELETE/REPLACE-Klausel verwenden.

Verwandte Statements

COMPRESS, SEPARATE.

Funktion

Das Statement EXAMINE dient dazu, den Inhalt eines alphanumerischen Feldes (oder eines Bereiches von Feldern innerhalb eines Arrays) nach einer bestimmten Zeichenkette abzusuchen; außerdem kann die Zeichenkette ausgetauscht werden, gezählt werden, wie oft sie vorkommt, und es können Informationen über die EXAMINE-Operation ausgegeben werden.

operand1

Operand1 ist das Feld, dessen Inhalt untersucht werden soll.

Ist *operand1* eine DYNAMISCHE Variable, kann deren Länge über eine REPLACE-Operation auf einen höheren oder niedrigeren Wert gesetzt werden; durch eine DELETE-Operation kann deren Länge auf "0" gesetzt werden. Die aktuelle Länge einer DYNAMISCHEN Variablen kann über die Systemvariable *LENGTH ermittelt werden. Sie finden allgemeine Informationen über DYNAMISCHE Variablen in der *Natural User's Documentation* sowie des weiteren im *Natural Benutzerhandbuch*.

operand4

Operand4 ist der für die Untersuchung zu verwendende Wert.

FULL

Wenn Sie für einen Operanden FULL angeben, so wird der gesamte Wert, einschließlich nachfolgender Leerstellen, verarbeitet; ohne FULL werden dem Wert nachfolgende Leerstellen bei der Verarbeitung ignoriert.

SUBSTRING

Normalerweise wird der ganze Inhalt des Feldes untersucht, und zwar vom Anfang des Feldes bis zum Ende bzw. bis zum letzten signifikanten Zeichen.

Die Option SUBSTRING ermöglicht es Ihnen, nur einen bestimmten Teil des Feldes zu untersuchen. In der SUBSTRING-Klausel geben Sie nach dem Feldnamen (*operand1*) zunächst die erste Stelle (*operand2*) und dann die Länge (*operand3*) des Feldteils, der untersucht werden soll, an.

Um zum Beispiel die 5. bis einschließlich 12. Stelle eines Feldes #A zu untersuchen, würden Sie folgendes angeben:

```
EXAMINE SUBSTRING(#A,5,8).
```

Anmerkung:

Wenn Sie operand2 weglassen, wird ab Anfang des Feldes untersucht. Wenn Sie operand3 weglassen, wird ab der angegebenen Stelle (operand2) bis zum Ende des Feldes untersucht.

PATTERN

Wenn Sie das Feld nach einem Wert absuchen möchten, der Variablen enthält, d.h. Platzhalter für Stellen, die bei der Suche nicht berücksichtigt werden sollen, verwenden Sie die Option **PATTERN**. *Operand4* kann dann die folgenden Platzhalter für nicht zu untersuchende Stellen enthalten:

- Ein Punkt (.), Fragezeichen (?) oder Unterstrich (_) steht für eine einzelne Stelle, die nicht untersucht werden soll.
- Ein Stern (*) oder Prozentzeichen (%) steht für eine beliebige Anzahl von Stellen, die nicht untersucht werden sollen.

Beispiel: Mit **PATTERN** 'NAT*AL' könnten Sie ein Feld nach jedem Wert, in dem "NAT" und "AL" vorkommen, absuchen, ganz gleich, welche und wieviele andere Zeichen zwischen "NAT" und "AL" stehen (dies würde z.B. auf die Werte NATURAL und NATIONAL zutreffen, aber auch auf NATAL).

DELIMITERS-option

{
ABSOLUTE
 WITH [DELIMITERS]
 WITH [DELIMITERS] *operand5*
 }

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand5	C S	A	ja	nein

Standardmäßig gilt die Option **ABSOLUTE**; d.h. die zu suchende Zeichenkette wird auch gefunden, wenn sie von anderen Zeichen umgeben und Teil einer längeren Zeichenkette ist.

Mit **WITH DELIMITERS** wird ein Wert gesucht, dem je ein Leerzeichen oder irgendein anderes Zeichen, das weder ein Buchstabe noch eine Ziffer ist, vor- und nachgestellt ist.

Mit **WITH DELIMITERS** *operand5* wird ein Wert gesucht, der von dem/den in *operand5* angegebenen Zeichen eingegrenzt ist.

DELETE/REPLACE-clause

$[\text{AND}] \left\{ \begin{array}{l} \text{DELETE } [\text{FIRST}] \\ \text{REPLACE } [\text{FIRST}] [\text{WITH}] \left[\text{FULL } [\text{VALUE } [\text{OF}]] \right] \text{ operand6} \end{array} \right\}$

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand6	C S	A	ja	nein

Mit der Option DELETE wird ein Wert aus *operand1* gelöscht.

Mit der Option REPLACE wird ein Wert durch den mit *operand6* angegebenen Wert ersetzt.

Das Schlüsselwort FIRST bewirkt, daß nur der erste vorkommende identische Wert gelöscht bzw. ersetzt wird.

Wenn der aus der REPLACE-Operation resultierende Wert zu lang ist und nicht in *operand1* paßt, erhalten Sie eine entsprechende Fehlermeldung.

Ist *operand1* eine DYNAMISCHE Variable, kann deren Länge über eine REPLACE-Operation auf einen höheren oder niedrigeren Wert gesetzt werden; durch eine DELETE-Operation kann deren Länge auf "0" gesetzt werden. Die aktuelle Länge einer DYNAMISCHEN Variablen kann über die Systemvariable *LENGTH ermittelt werden. Sie finden allgemeine Informationen über DYNAMISCHE Variablen in der *Natural User's Documentation* sowie des weiteren im *Natural Benutzerhandbuch*.

GIVING-clause

[GIVING]	$\left[\begin{array}{c} \text{NUMBER} \\ \text{POSITION} \\ \text{LENGTH} \\ \text{INDEX} \end{array} \right]$	[IN] <i>operand7</i>
----------	---	----------------------

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand7	S	N P I	ja	ja

Mit **GIVING NUMBER** erhalten Sie die Zahl, wie oft der zu suchende Wert vorkommt. Wenn Sie gleichzeitig REPLACE FIRST oder DELETE FIRST verwenden, erhalten Sie maximal die Zahl 1.

Mit **GIVING POSITION** erhalten Sie die Byte-Position, die der erste gefundene Wert (*operand4*) innerhalb von *operand1* (bzw. des Substrings von *operand1*) innehat.

Mit **GIVING LENGTH** erhalten Sie die Länge von *operand1* (bzw. des Substrings von *operand1*), nachdem alle DELETE- bzw. REPLACE-Operationen abgeschlossen sind.

GIVING INDEX

[GIVING] INDEX [IN] <i>operand7...3</i>

Mit GIVING INDEX erhalten Sie die Nummer der Ausprägung (Index) von *operand1*, in der der erste gefundene Wert (*operand4*) enthalten ist.

GIVING INDEX ist nur möglich, wenn *operand1* ein Array ist. *Operand7* muß genauso oft angegeben werden wie *operand1* Dimensionen hat (höchstens dreimal).

Operand7 enthält "0", wenn der gesuchte Wert in keiner der Ausprägungen enthalten ist.

Anmerkung:

Falls der Indexbereich von *operand1* die Ausprägung 0 enthält (z.B. 0:5), ist der Wert "0" in *operand7* zweideutig. In diesem Falle sollte eine zusätzliche GIVING NUMBER-Klausel verwendet werden, um festzustellen, ob der gesuchte Wert tatsächlich vorkommt oder nicht.

Beispiel 1

```

/* EXAMPLE 'EXMEX1': EXAMINE
/*****
DEFINE DATA LOCAL
1 #TEXT (A40)
1 #A (A1)
1 #NMB1 (N2)
1 #NMB2 (N2)
1 #NMB3 (N2)
1 #NMBEX2 (N2)
1 #NMBEX3 (N2)
1 #NMBEX4 (N2)
1 #POSEX5 (N2)
1 #LGHEX6 (N2)
END-DEFINE
/*****
WRITE 'EXAMPLE 1 (GIVING NUMBER, WITH DELIMITER)'
MOVE 'ABC A B C .A. .B. .C. -A- -B-' TO #TEXT
ASSIGN #A = 'A'
EXAMINE #TEXT FOR #A GIVING NUMBER #NMB1
EXAMINE #TEXT FOR #A WITH DELIMITER GIVING NUMBER #NMB2
EXAMINE #TEXT FOR #A WITH DELIMITER '.' GIVING NUMBER #NMB3
WRITE NOTITLE '=' #NMB1 '=' #NMB2 '=' #NMB3
/*****
WRITE / 'EXAMPLE 2 (WITH DELIMITER, REPLACE, GIVING NUMBER)'
WRITE '=' #TEXT
EXAMINE #TEXT FOR #A WITH DELIMITER '-' REPLACE WITH '*'
GIVING NUMBER #NMBEX2
WRITE '=' #TEXT '=' #NMBEX2
/*****
WRITE / 'EXAMPLE 3 (REPLACE, GIVING NUMBER)'
WRITE '=' #TEXT
EXAMINE #TEXT ' ' REPLACE WITH '+' GIVING NUMBER #NMBEX3
WRITE '=' #TEXT '=' #NMBEX3
/*****
WRITE / 'EXAMPLE 4 (FULL, REPLACE, GIVING NUMBER)'
WRITE '=' #TEXT
EXAMINE FULL #TEXT ' ' REPLACE WITH '+' GIVING NUMBER #NMBEX4
WRITE '=' #TEXT '=' #NMBEX4
/*****
WRITE / 'EXAMPLE 5 (DELETE, GIVING POSITION)'
WRITE '=' #TEXT
EXAMINE #TEXT '+' DELETE GIVING POSITION #POSEX5
WRITE '=' #TEXT '=' #POSEX5
/*****
WRITE / 'EXAMPLE 6 (DELETE, GIVING LENGTH)'
WRITE '=' #TEXT
EXAMINE #TEXT FOR 'A' DELETE GIVING LENGTH #LGHEX6
WRITE '=' #TEXT '=' #LGHEX6
END

```

EXAMPLE 1 (GIVING NUMBER, WITH DELIMITER)

#NMB1: 4 #NMB2: 3 #NMB3: 1

EXAMPLE 2 (WITH DELIMITER, REPLACE, GIVING NUMBER)

#TEXT: ABC A B C .A. .B. .C. -A- -B-

#TEXT: ABC A B C .A. .B. .C. -* -B- #NMBEX2: 1

EXAMPLE 3 (REPLACE, GIVING NUMBER)

#TEXT: ABC A B C .A. .B. .C. -* -B-

#TEXT: ABC+++A+B+C+++ .A.++.B.++.C.++++-*---B- #NMBEX3: 18

EXAMPLE 4 (FULL, REPLACE, GIVING NUMBER)

#TEXT: ABC+++A+B+C+++ .A.++.B.++.C.++++-*---B-

#TEXT: ABC+++A+B+C+++ .A.++.B.++.C.++++-*---B-+ #NMBEX4: 1

EXAMPLE 5 (DELETE, GIVING POSITION)

#TEXT: ABC+++A+B+C+++ .A.++.B.++.C.++++-*---B-

#TEXT: ABCABC.A..B..C.-*---B- #POSEX5: 4

EXAMPLE 6 (DELETE, GIVING LENGTH)

#TEXT: ABCABC.A..B..C.-*---B-

#TEXT: BCBC...B..C.-*---B- #LGHEX6: 18

Beispiel 2

```

/* EXAMPLE 'EXMEX2': EXAMINE SUBSTRING, PATTERN, TRANSLATE
/*****
DEFINE DATA LOCAL
1 #TEXT (A50)
1 #A (A7)
1 #NMB (N2)
1 #START (N2)
1 #TAB(A2/1:10)
END-DEFINE
/*****
MOVE 'ABC  A B C  .A.  .B.  .C.  -A-  -B-  -C-' TO #TEXT
/*****
ASSIGN #A = 'A B C'
ASSIGN #START = 6
EXAMINE SUBSTRING(#TEXT,#START,9) FOR #A GIVING NUMBER #NMB
WRITE NOTITLE '=' #NMB
/*****
EXAMINE #TEXT FOR PATTERN '*B' GIVING NUMBER #NMB
WRITE NOTITLE '=' #NMB
/*****
MOVE 'AX' TO #TAB(1)
MOVE 'BY' TO #TAB(2)
MOVE 'CZ' TO #TAB(3)
EXAMINE #TEXT TRANSLATE USING #TAB(*)
WRITE NOTITLE '=' #TEXT
EXAMINE #TEXT TRANSLATE USING INVERTED #TAB(*)
WRITE NOTITLE '=' #TEXT
/*****
END

```

#NMB:	1								
#NMB:	4								
#TEXT:	XYZ	X Y Z	.X.	.Y.	.Z.	-X-	-Y-	-Z-	
#TEXT:	ABC	A B C	.A.	.B.	.C.	-A-	-B-	-C-	

EXAMINE TRANSLATE

$$\text{EXAMINE} \left\{ \begin{array}{l} \text{SUBSTRING} \begin{array}{l} \textit{operand1} \\ (\textit{operand1}, \textit{operand2}, \textit{operand3}) \end{array} \\ \text{TRANSLATE} \left\{ \begin{array}{l} \text{INTO} \left\{ \begin{array}{l} \text{UPPER} \\ \text{LOWER} \end{array} \right\} [\text{CASE}] \\ \text{USING} [\text{INVERTED}] \textit{operand4} \end{array} \right\} \end{array} \right\} [\text{AND}]$$

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S A	A	ja	nein
Operand2	C S	N P I	ja	nein
Operand3	C S	N P I	ja	nein
Operand4	S A	A B	ja	nein

Funktion

Das Statement EXAMINE TRANSLATE dient dazu, die in einem Feld enthaltenen Zeichen in Groß- oder Kleinschreibung oder in andere Zeichen umzusetzen.

operand1

Operand1 ist das Feld, dessen Inhalt umgesetzt werden soll.

SUBSTRING

Normalerweise wird der Inhalt des gesamten Feldes umgesetzt.

Die Option SUBSTRING ermöglicht es Ihnen, nur einen bestimmten Teil des Feldes umzusetzen. In der SUBSTRING-Klausel geben Sie nach dem Feldnamen (*operand1*) zunächst die erste Stelle (*operand2*) und dann die Länge (*operand3*) des Feldteils, der umgesetzt werden soll, an.

Um zum Beispiel die 5. bis einschließlich 12. Stelle eines Feldes #A umzusetzen, würden Sie folgendes angeben:

```
EXAMINE SUBSTRING(#A,5,8) AND TRANSLATE ...
```

Anmerkung:

Wenn Sie operand2 weglassen, wird ab Anfang des Feldes umgesetzt. Wenn Sie operand3 weglassen, wird ab der angegebenen Stelle (operand2) bis zum Ende des Feldes umgesetzt.

INTO UPPER/LOWER CASE

Wenn Sie INTO UPPER CASE angeben, wird der Inhalt von *operand1* in Großbuchstaben umgesetzt.

Wenn Sie INTO LOWER CASE angeben, wird der Inhalt von *operand1* in Kleinbuchstaben umgesetzt.

Umsetzungstabelle

Operand4 ist die Umsetzungstabelle, die für die Zeichenumsetzung verwendet werden soll.

Die Tabelle muß Format/Länge A2 oder B2 haben.

Anmerkung:

Falls für ein umzusetzendes Zeichen in der Umsetzungstabelle mehr als eine Umsetzung definiert ist, gilt die jeweils letzte Umsetzung.

INVERTED

Wenn Sie das Schlüsselwort INVERTED angeben, wird die Umsetzungstabelle (*operand4*) in umgekehrter Richtung verwendet, d.h. die Umsetzungsrichtung wird umgekehrt.

Beispiel

```

/* EXAMPLE 'EXMEX2': EXAMINE SUBSTRING, PATTERN, TRANSLATE
/*****
DEFINE DATA LOCAL
1 #TEXT (A50)
1 #A (A7)
1 #NMB (N2)
1 #START (N2)
1 #TAB(A2/1:10)
END-DEFINE
/*****
MOVE 'ABC  A B C  .A.  .B.  .C.  -A-  -B-  -C-' TO #TEXT
/*****
ASSIGN #A = 'A B C'
ASSIGN #START = 6
EXAMINE SUBSTRING(#TEXT,#START,9) FOR #A GIVING NUMBER #NMB
WRITE NOTITLE '=' #NMB
/*****
EXAMINE #TEXT FOR PATTERN '*B' GIVING NUMBER #NMB
WRITE NOTITLE '=' #NMB
/*****
MOVE 'AX' TO #TAB(1)
MOVE 'BY' TO #TAB(2)
MOVE 'CZ' TO #TAB(3)
EXAMINE #TEXT TRANSLATE USING #TAB(*)
WRITE NOTITLE '=' #TEXT
EXAMINE #TEXT TRANSLATE USING INVERTED #TAB(*)
WRITE NOTITLE '=' #TEXT
/*****
END

```

#NMB:	1								
#NMB:	4								
#TEXT:	XYZ	X Y Z	.X.	.Y.	.Z.	-X-	-Y-	-Z-	
#TEXT:	ABC	A B C	.A.	.B.	.C.	-A-	-B-	-C-	

EXPAND

Anmerkung:

Dieses Statement steht auf Großrechnern nicht zur Verfügung.

EXPAND [SIZE OF] DYNAMIC [VARIABLE] *operand1* TO *operand2*

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S	A B	nein	nein
Operand2	C S	I	nein	nein

Verwandte Statements

REDUCE.

Funktion

Mit dem Statement EXPAND DYNAMIC VARIABLE wird die zugewiesene Länge einer dynamischen Variablen (*operand1*) auf den mit dem in *operand2* angegebenen Wert erweitert. Wenn *operand2* kleiner als die aktuell zugewiesene Länge von *operand1* ist, wird das Statement für diese dynamische Variable ignoriert. Die aktuell verwendete Länge (*LENGTH) der dynamischen Variablen wird nicht geändert.

operand1

Operand1 ist die dynamische Variable, für die die Länge erweitert werden soll.

operand2

Operand2 dient dazu, die neue Länge anzugeben. Der angegebene Wert muß ein nicht negativer numerischer Wert sein.

FETCH

FETCH REPEAT
RETURN *operand1* [*operand2* [(*parameter*)]] ...

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	ja	nein
Operand2	C S A G	A N P I F B D T L G	ja	ja

Verwandte Statements

CALLNAT, PERFORM.

Funktion

Das Statement FETCH dient dazu, ein Natural-Objektprogramm auszuführen, welches als Hauptprogramm geschrieben wurde. Das zu ladende Programm muß vorher mit STOW oder CATALOG in der Natural-Systemdatei in Objektform gespeichert worden sein. Ein im Arbeitsbereich des Editors befindliches Sourceprogramm wird durch die Ausführung eines FETCH-Statements nicht überlagert.

REPEAT

REPEAT bewirkt, daß bei der Ausführung des aufgerufenen Programms der Benutzer bei INPUT-Statements keine Eingaben machen muß. REPEAT kann auch dazu eingesetzt werden, Informationen über die Ausführung des Programms am Bildschirm anzuzeigen, ohne daß der Benutzer daraufhin EINGABE drücken muß.

RETURN

Wird RETURN nicht angegeben, so wird das aufrufende Programm, welches das FETCH-Statement enthält, augenblicklich beendet, und das aufgerufene Programm wird als *Hauptprogramm* (Stufe 1) aktiviert.

Geben Sie FETCH RETURN an, wird das aufrufende Programm nicht beendet, sondern nur unterbrochen, während das aufgerufene Programm als *Unterprogramm* auf einer höheren Stufe ausgeführt wird. Ein END- oder ESCAPE ROUTINE-Statement im aufgerufenen Programm bewirkt, daß die Kontrolle wieder an das aufrufende Programm übergeben wird, dessen Ausführung dann mit dem auf das FETCH RETURN folgende Statement fortgesetzt wird.

FETCH RETURN ermöglicht es, ein Objekt vom Typ Programm aufzurufen und als Unterprogramm auszuführen.

Programmname (*operand1*)

Der Name des aufgerufenen Programms (maximal 8 Zeichen lang) kann entweder als alphanumerische Konstante oder als Inhalt einer alphanumerischen Variablen der Länge 1 bis 8 angegeben werden.

Natural sucht das Programm zunächst in der zum Zeitpunkt der Ausführung des FETCH-Statements gerade aktiven Library. Wird es dort nicht gefunden, sucht Natural in den Steplib. Wird das Programm auch dort nicht gefunden, gibt Natural eine entsprechende Fehlermeldung aus.

Der Name des Programms darf ein Und-Zeichen (&) enthalten; zur Laufzeit wird dieses Zeichen durch den aktuellen Wert der Systemvariablen *LANGUAGE ersetzt. Dadurch ist es beispielsweise möglich, je nachdem in welcher Sprache eine Eingabe gemacht wird, zur Verarbeitung der Eingabe unterschiedliche Programme aufzurufen.

Parameter (*operand2*)

Mit dem FETCH-Statement können auch Parameterfelder an das aufgerufene Programm übergeben werden. Ein Parameterfeld kann mit einem beliebigen Format definiert werden; das Format wird dem jeweiligen INPUT-Feld entsprechend umgesetzt. Sämtliche Parameter werden oben auf dem Natural-Stack abgelegt.

Das aufgerufene Programm liest die übergebenen Parameterfelder über ein INPUT-Statement. Das erste INPUT-Statement bewirkt, daß die Werte aller Parameterfelder in die mit dem INPUT-Statement angegebenen Felder übertragen werden. Da jedes mit einem numerischen Format definierte Parameterfeld eine Stelle für das Vorzeichen erhält, wenn sein Wert negativ ist, muß beim INPUT-Statement der Session-Parameter SG für die Parameterfelder auf SG=ON gesetzt werden.

Werden mehr Parameter übergeben als vom INPUT-Statement gelesen werden können, so werden überschüssige Parameter ignoriert. Die Anzahl der Parameter kann mit Hilfe der Natural-Systemvariablen *DATA ermittelt werden.

Anmerkung:

Wenn operand2 eine Zeitvariable (Format T) ist, wird nur die Zeitkomponente des Variableninhalts übergeben, aber nicht die Datumskomponente.

parameter

Wenn *operand2* eine Datumvariable ist, können Sie den Session-Parameter DF als *parameter* für diese Variable angeben. Der Session-Parameter DF ist im *Natural Referenzhandbuch* beschrieben.

Weitere Hinweise

Zusätzlich zu den explizit mit dem FETCH-Statement übergebenen Parametern hat das aufgerufene Programm Zugang zu der Global Data Area des aufrufenden Programms.

Je nachdem, wie Ihr Natural-Administrator den Natural-Profilparameter OPRB gesetzt hat, kann es sein, daß das FETCH-Statement die Ausführung eines internen END TRANSACTION-Statements auslöst. Soll eine logische Transaktion mehrere Programme einschließen, so wenden Sie sich bitte vorher an Ihren Natural-Administrator, um sicherzustellen, daß der OPRB-Parameter entsprechend gesetzt ist.

Beispiel

Aufrufendes Programm:

```

/* EXAMPLE 'FETEX1': FETCH
/*****
DEFINE DATA LOCAL
1 #PNUM (A8)
1 #FNC (A1)
END-DEFINE
/*****
INPUT 10X 'SELECTION MENU FOR EMPLOYEES SYSTEM' /
      10X '-' (54) //
      10X 'ADD ' '(A)' /
      10X 'PURGE' '(P)' /
      10X 'UPDATE' '(U)' /
      10X 'TERMINATE' '(.)' //
      10X 'PERSONNEL NUMBER:' #PNUM ///
      10X 'PLEASE ENTER FUNCTION: ' #FNC
/*****
DECIDE ON EVERY VALUE OF #FNC
  VALUE 'A'
    FETCH 'ADD-RT' #PNUM
  VALUE 'P'
    FETCH 'PUR-RT' #PNUM
  VALUE 'U'
    FETCH 'UPD-RT' #PNUM
  VALUE '.'
    STOP
  NONE
  REINPUT 'PLEASE ENTER A VALID FUNCTION' MARK *#FNC
END-DECIDE
/*****
END

```

Aufgerufenes Programm:

```

/* EXAMPLE 'PUR-RT' (PROGRAM FETCHED IN EXAMPLE 'FETEX1')
/*****
DEFINE DATA LOCAL
  1 #PERS-NR (A8)
  1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
END-DEFINE
/*****
INPUT #PERS-NR
/*****
FIND NUMBER EMPLOY-VIEW WITH PERSONNEL-ID = #PERS-NR
  IF *NUMBER = 0
    WRITE NOTITLE 'NO RECORD FOUND'
    STOP
  END-IF
/*****
FIND EMPLOY-VIEW WITH PERSONNEL-ID = #PERS-NR
  DELETE
  END TRANSACTION
  WRITE NOTITLE 'RECORD DELETED'
END-FIND
/*****
END

```

SELECTION MENU FOR EMPLOYEES SYSTEM

ADD (A)
 PURGE (P)
 UPDATE (U)
 TERMINATE (.)

PERSONNEL NUMBER: 1150304

PLEASE ENTER FUNCTION: P

RECORD DELETED

FIND

FIND	[ALL (operand1) FIRST NUMBER UNIQUE]	[RECORDS] [IN] [FILE] <i>view-name</i>
		[PASSWORD=operand2] [CIPHER=operand3] [[WITH] [[LIMIT] (operand4)] <i>basic-search-criterion</i> [COUPLED-clause]... ^{4/42} [STARTING WITH ISN=operand5] [SORTED BY-clause] [RETAIN-clause] [WHERE-clause] [IF NO RECORDS FOUND-clause] <i>statement...</i>
END-FIND	<i>(nur im Structured Mode)</i>	
[LOOP]	<i>(nur im Reporting Mode)</i>	

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	N P I	ja	nein
Operand2	C S	A	ja	nein
Operand3	C S	N	ja	nein
Operand4	C S	N P I B	ja	nein
Operand5	C S	N P I B	ja	nein

Verwandte Statements

READ, HISTOGRAM.

Funktion

Das Statement FIND dient dazu, Datensätze von der Datenbank auszuwählen, und zwar anhand eines Suchkriteriums, d.h. des Wertes eines Schlüsselfeldes (Deskriptors).

Mit dem FIND-Statement wird eine Verarbeitungsschleife initiiert, die für jeden gefundenen Datensatz durchlaufen wird. Innerhalb der FIND-Schleife kann jedes Feld eines gefundenen Datensatzes referenziert werden, und zwar ohne daß hierzu ein zusätzliches READ-Statement erforderlich wäre.

Hinweise für DL/I-Datenbanken

Beim Zugriff auf ein Feld, das nach dem letzten Byte eines bestimmten Segmentvorkommens beginnt, wird die Speicherkopie dieses Feldes entsprechend ihres Formats (numerisch, leer, usw.) gefüllt. Der DL/I-Begriff *Segmentvorkommen* (segment occurrences) sollte im Sinne des hier verwendeten Begriffs *Datensätze* (records) verstanden werden, wie er in dieser Beschreibung des FIND-Statements verwendet wird.

Hinweise für SQL-Datenbanken

FIND FIRST sowie die PASSWORD-, CIPHER-, COUPLED- und RETAIN-Klauseln sind nicht erlaubt.

FIND UNIQUE ist nicht erlaubt. (Ausnahme: auf Großrechnern kann FIND UNIQUE für Primärschlüssel verwendet werden; allerdings ist dies nur aus Kompatibilitätsgründen erlaubt und sollte nicht benutzt werden.)

Die SORTED BY-Klausel entspricht der SQL-Klausel ORDER BY.

Das Suchkriterium (*basic-search-criterion*) für eine SQL-Datenbank-Tabelle kann genauso angegeben werden wie für eine Adabas-Datei. Der Begriff *Datensatz* ist in diesem Zusammenhang dem SQL-Begriff *Reihe* gleichzusetzen.

Hinweise für VSAM-Datenbanken

Das FIND-Statement ist nur auf VSAM-KSDS (key-sequenced datasets) und -ESDS (entry-sequenced datasets) anwendbar. Bei ESDS muß ein Alternativ-Index für den Basis-Cluster definiert werden.

Einschränkungen für Entire System Server

FIND NUMBER und FIND UNIQUE sowie die PASSWORD-, CIPHER-, COUPLED- und RETAIN-Klauseln sind nicht erlaubt.

Anzahl der Datensätze (*ALL/operand1*)

Sie können die Anzahl der ausgewählten Datensätze, die in der FIND-Schleife verarbeitet werden sollen, auf eine bestimmte Zahl begrenzen. Diese Zahl (*operand1*) geben Sie in Klammern unmittelbar nach dem Wort FIND an, und zwar entweder als numerische Konstante oder in Form einer numerischen Variablen. Andernfalls werden alle gefundenen Datensätze weiterverarbeitet, was Sie zusätzlich durch das Schlüsselwort ALL betonen können.

Geben Sie mit *operand1* ein Limit an, so gilt dieses Limit für die initiierte FIND-Schleife, wobei allerdings Datensätze, die aufgrund einer WHERE-Klausel abgelehnt werden, nicht mitgezählt werden.

```
FIND (5) IN EMPLOYEES WITH ...
```

```
MOVE 10 TO #CNT(N2)  
FIND (#CNT) EMPLOYEES WITH ...
```

Das angegebene Limit hat für dieses Statement Vorrang vor einem mit einem LIMIT-Statement gesetzten Limit.

Ist mit dem LT-Parameter ein kleineres Limit gesetzt, so gilt das LT-Limit.

Anmerkungen:

Wenn Sie eine vierstellige Anzahl von Datensätzen verarbeiten möchten, geben Sie diese mit einer vorangestellten Null an: (0nnnn); denn Natural interpretiert jede vierstellige Zahl in Klammern als Zeilennummer-Referenzierung auf ein Statement.

Operand1 hat keinen Einfluß auf die Größe eines ISN-Set, der mittels einer RETAIN-Klausel erzeugt wird.

Operand1 wird zu Beginn des ersten FIND-Schleifendurchlaufs ausgewertet. Wird der Wert von operand1 innerhalb der FIND-Schleife geändert, hat dies keine Auswirkungen auf die Anzahl der verarbeiteten Datensätze.

FIND FIRST, FIND NUMBER, FIND UNIQUE

Diese Optionen dienen dazu, nur den ersten der gefundenen Datensätze auszuwählen (FIND FIRST), die Anzahl der gefundenen Datensätze zu ermitteln (FIND NUMBER) bzw. sicherzustellen, daß nur ein Datensatz das Suchkriterium erfüllt (FIND UNIQUE).

Näheres hierzu finden Sie in den entsprechenden Abschnitten gegen Ende der Beschreibung des FIND-Statements.

view-name

Der Name eines Views, der entweder in einem DEFINE DATA-Block oder in einer separaten Global oder Local Data Area definiert ist. Im *Reporting Mode* darf *view-name* auch der Name eines DDMs sein.

PASSWORD-Klausel

Die PASSWORD-Klausel gilt nur für Zugriffe auf Adabas- oder VSAM-Datenbanken. Mit Entire System Server ist diese Klausel nicht erlaubt.

Die PASSWORD-Klausel dient dazu, ein Paßwort (*operand2*) anzugeben, um auf Daten einer paßwort-geschützten Datei zugreifen zu können. Wollen Sie auf eine paßwort-geschützte Datei zugreifen, sollten Sie sich bezüglich des Paßwortes mit Ihrem Datenbank-Security-Administrator in Verbindung setzen.

Wird das Paßwort als Konstante angegeben, sollte die PASSWORD-Klausel ganz am Anfang einer Sourcecode-Zeile stehen; dadurch ist gewährleistet, daß das Paßwort im Sourcecode nicht sichtbar ist. Im TP-Modus können Sie die PASSWORD-Klausel unsichtbar machen, indem Sie zuerst das Terminalkommando “%*” eingeben, bevor Sie die PASSWORD-Klausel eintippen.

Wenn Sie die PASSWORD-Klausel weglassen, gilt das mit dem PASSW-Statement angegebene Paßwort.

Während der Ausführung einer Verarbeitungsschleife kann das Paßwort nicht geändert werden.

Beispiel für PASSWORD-Klausel:

```
/* EXAMPLE 'FNDPWD': FIND (USING PASSWORD CLAUSE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 PERSONNEL-ID
1 #PASS (A8)
END-DEFINE
/*****
INPUT 'ENTER PASSWORD FOR EMPLOYEE FILE:' #PASS (AD=N)
LIMIT 2
/*****
FIND EMPLOY-VIEW
  PASSWORD = #PASS
  WITH NAME = 'SMITH'
  DISPLAY NOTITLE NAME PERSONNEL-ID
END-FIND
/*****
END
```

ENTER PASSWORD FOR EMPLOYEE FILE:

CIPHER-Klausel

Die CIPHER-Klausel gilt nur für Zugriffe auf Adabas-Dateien. Mit Entire System Server ist diese Klausel nicht erlaubt.

Die CIPHER-Klausel dient dazu, einen Cipher-Code (Chiffrierschlüssel) anzugeben (*operand3*), um in chiffrierter Form gespeicherte Daten von Adabas-Dateien in entschlüsselter Form zu erhalten. Wollen Sie auf eine chiffrierte Datei zugreifen, sollten Sie sich bezüglich des Chiffrierschlüssels mit Ihrem Datenbank-Security-Administrator in Verbindung setzen.

Der Chiffrierschlüssel kann als numerische Konstante (8 Stellen lang) oder als Inhalt einer Benutzervariablen (Format/Länge N8) angegeben werden.

Wird der Chiffrierschlüssel als Konstante angegeben, sollte die CIPHER-Klausel ganz am Anfang einer Sourcecode-Zeile stehen; dadurch ist gewährleistet, daß der Chiffrierschlüssel im Sourcecode nicht sichtbar ist. Im TP-Modus können Sie die CIPHER-Klausel unsichtbar machen, indem Sie zuerst das Terminalkommando “%*” eingeben, bevor Sie die CIPHER-Klausel eintippen.

Während der Ausführung der FIND-Verarbeitungsschleife kann der Chiffrierschlüssel nicht geändert werden.

Beispiel für CIPHER-Klausel:

```

/* EXAMPLE 'FNDCIP': FIND (USING CIPHER CLAUSE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 PERSONNEL-ID
1 #PASS (A8)
1 #CIPHER (N8)
END-DEFINE
/*****
LIMIT 2
INPUT 'ENTER PASSWORD FOR EMPLOYEE FILE: ' #PASS (AD=N)
  / 'ENTER CIPHER KEY FOR EMPLOYEE FILE: ' #CIPHER (AD=N)
/*****
FIND EMPLOY-VIEW
  PASSWORD = #PASS
  CIPHER = #CIPHER
  WITH NAME = 'SMITH'
  DISPLAY NOTITLE NAME PERSONNEL-ID
END-FIND
/*****
END

```

```

ENTER PASSWORD FOR EMPLOYEE FILE:
ENTER CIPHER KEY FOR EMPLOYEE FILE:

```

WITH-Klausel

Die WITH-Klausel ist unbedingt erforderlich. Mit ihr wird das Suchkriterium (*basic-search-criterion*) in Form des Wertes eines als Schlüsselfeld (Deskriptor) definierten Datenbankfeldes angegeben.

Bei Adabas-Dateien kann als Schlüsselfeld in der WITH-Klausel ein Deskriptor, Subdeskriptor, Superdeskriptor, Hyperdeskriptor oder phonetischer Deskriptor angegeben werden. Auf Großrechnern kann auch ein Nicht-Deskriptor (d.h. ein Feld, das im DDM mit “N” markiert ist) angegeben werden.

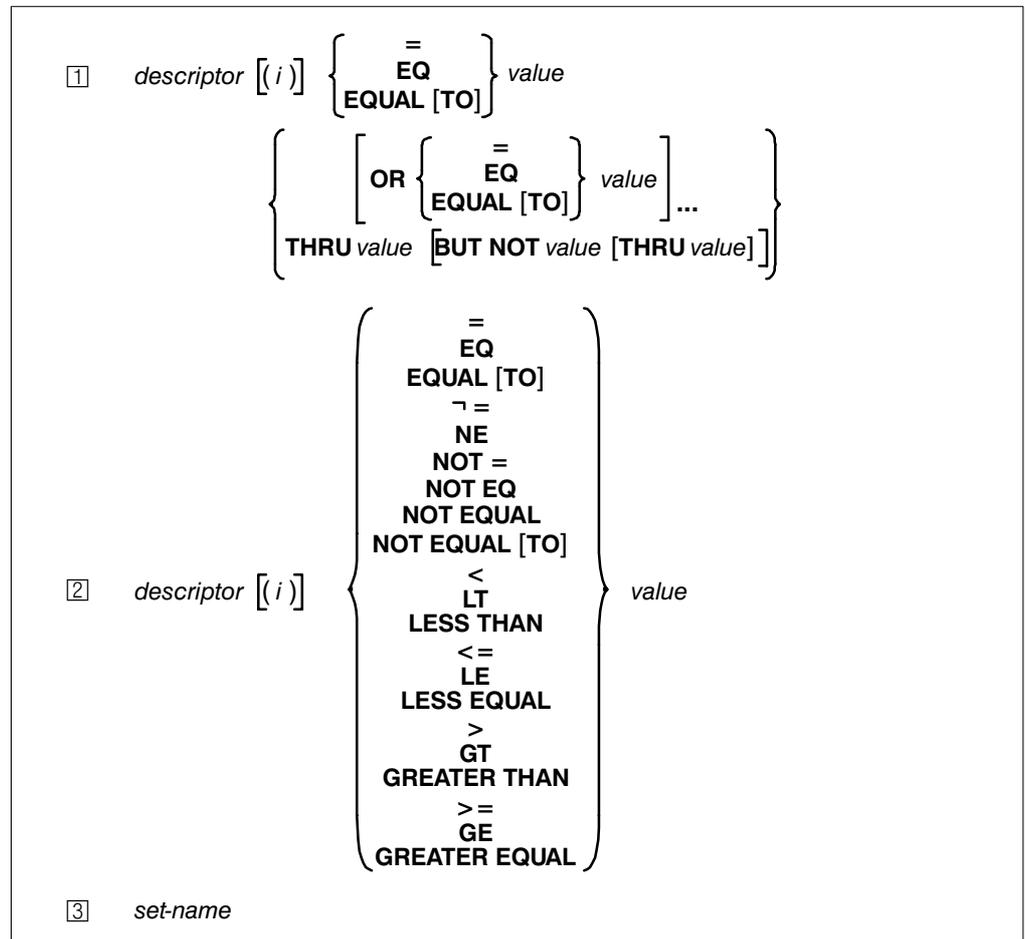
Bei DL/I-Dateien kann nur ein im DDM mit “D” markiertes Schlüsselfeld angegeben werden.

Bei VSAM-Dateien kann nur ein VSAM-Schlüsselfeld (key field) angegeben werden.

Die Anzahl der Datensätze, die anhand des in der WITH-Klausel definierten Suchkriteriums ausgewählt werden sollen, können Sie begrenzen, indem Sie das Schlüsselwort LIMIT und dahinter in Klammern eine Zahl oder eine Benutzervariable angeben (*operand4*). Übersteigt die Anzahl der ausgewählten Datensätze diese Zahl, wird das Programm mit einer entsprechenden Fehlermeldung abgebrochen.

Anmerkung:

Wenn das Limit eine vierstellige Zahl sein soll, geben Sie diese mit einer vorangestellten Null an: (0nnnn); denn Natural interpretiert jede vierstellige Zahl in Klammern als Zeilennummer-Referenzierung auf ein Statement.

Suchkriterium für Adabas-Dateien (*basic-search-criterion*)

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Descriptor	S A	A N P I F B D T L	nein	nein
Value	C S	A N P I F B D T L	ja	nein
Set-name	C S	A	nein	nein

descriptor

Es kann ein Adabas-Deskriptor, -Subdeskriptor, -Superdeskriptor, -Hyperdeskriptor oder phonetischer Deskriptor angegeben werden. Es kann auch ein im DDM als Nicht-Deskriptor markiertes Feld angegeben werden.

i

Index. Ein Deskriptorfeld, das Teil einer Periodengruppe ist, kann mit oder ohne Index angegeben werden. Wird kein Index angegeben, so wird ein Datensatz ausgewählt, wenn der Suchwert in einer beliebigen Ausprägung gefunden wird. Wird ein Index angegeben, so wird ein Datensatz nur ausgewählt, wenn der Suchwert in der im Index angegebenen Ausprägung gefunden wird. Der Index muß als Konstante angegeben werden; es darf kein Indexbereich angegeben werden.

Ist das angegebene Deskriptorfeld ein multiples Feld, darf kein Index angegeben werden. Ein Datensatz wird ausgewählt unabhängig davon, in welcher Ausprägung des Feldes der Suchwert gefunden wird.

value

Der Suchwert, den das Deskriptorfeld haben soll. Die Formate von Suchwert und Deskriptorfeld müssen kompatibel sein.

set-name

Identifiziert einen Set von Datensätzen, die mit einem FIND-Statement ausgewählt wurden, das eine RETAIN-Klausel enthielt. Der referenzierte Set muß von derselben physischen Adabas-Datei ausgewählt worden sein. *set-name* kann entweder als Textkonstante (bis zu 32 Zeichen lang) oder in Form einer alphanumerischen Variablen angegeben werden. Mit Entire System Server kann *set-name* nicht angegeben werden.

Beispiele für Basic Search Criterion in WITH-Klausel:

```

FIND STAFF WITH NAME = 'SMITH'

FIND STAFF WITH CITY NE 'BOSTON'

FIND STAFF WITH BIRTH = 19610803

FIND STAFF WITH BIRTH = 19610803 THRU 19610811

FIND STAFF WITH NAME = 'O HARA' OR = 'JONES' OR = 'JACKSON'

FIND STAFF WITH PERSONNEL-ID = 100082 THRU 100100
                                BUT NOT 100087 THRU 100095

```

Beispiele für Basic Search Criterion mit multiplem Feld:

Wenn in einem *basic-search-criterion* als Deskriptor ein multiples Feld verwendet wird, lassen sich grundsätzlich vier verschiedene Arten von Ergebnissen erzielen (die folgenden Beispiele gehen davon aus, daß das Feld MU-FIELD ein multiples Feld ist):

1. FIND XYZ-VIEW WITH **MU-FIELD = 'A'**

Dieses Statement liefert Datensätze, in denen *mindestens eine* Ausprägung von MU-FIELD den Wert "A" enthält.

2. FIND XYZ-VIEW WITH **MU-FIELD NOT EQUAL 'A'**

Dieses Statement liefert Datensätze, in denen *mindestens eine* Ausprägung von MU-FIELD *nicht* den Wert "A" enthält.

3. FIND XYZ-VIEW WITH **NOT MU-FIELD NOT EQUAL 'A'**

Dieses Statement liefert Datensätze, in denen *jede* Ausprägung von MU-FIELD den Wert "A" enthält.

4. FIND XYZ-VIEW WITH **NOT MU-FIELD = 'A'**

Dieses Statement liefert Datensätze, in denen *keine* der Ausprägungen von MU-FIELD den Wert "A" enthält.

Suchkriterium mit Null-Indikator (*basic-search-criterion*)

$$\text{null-indicator} \left\{ \begin{array}{c} = \\ \text{EQ} \\ \text{EQUAL [TO]} \end{array} \right\} \text{value}$$

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Null-indicator	S	I	nein	nein
Value	C S	N P I F B	ja	nein

Möglicher Wert (*value*) ist “-1” (= das betreffende Feld enthält keinen Wert) oder “0” (= das betreffende Feld enthält einen Wert).

Verknüpfen von Suchkriterien (für Adabas-Dateien)

Es ist möglich, mehrere Suchkriterien mit den Boole'schen Operatoren AND, OR und NOT miteinander zu verknüpfen. Mit Klammern kann außerdem die Reihenfolge der Kriterienauswertung gesteuert werden. Die Auswertung verknüpfter Suchkriterien geschieht in folgender Reihenfolge:

- ① () Klammern
- ② **NOT** Negation (nur für *basic-search-criterion* der Form ②).
- ③ **AND** "Und"-Verknüpfung
- ④ **OR** "Oder"-Verknüpfung

Basic-search-criteria können mit logischen Operatoren verknüpft werden, um einen komplexen Suchausdruck (*search-expression*) zu bilden. Eine solche komplexe *search-expression* hat folgende Syntax:

$$[\text{NOT}] \left\{ \begin{array}{l} \textit{basic-search-criterion} \\ \textit{search-expression} \end{array} \right\} \left[\left\{ \begin{array}{l} \text{OR} \\ \text{AND} \end{array} \right\} \textit{search-expression} \right] \dots$$

Beispiele für komplexen Suchausdruck in WITH-Klausel:

```
FIND STAFF WITH BIRTH LT 19770101 AND DEPT = 'DEPT06'
```

```
FIND STAFF WITH JOB-TITLE = 'CLERK TYPIST'
AND (BIRTH GT 19560101 OR LANG = 'SPANISH')
```

```
FIND STAFF WITH JOB-TITLE = 'CLERK TYPIST'
AND NOT (BIRTH GT 19560101 OR LANG = 'SPANISH')
```

```
FIND STAFF WITH DEPT = 'ABC' THRU 'DEF'
AND CITY = 'WASHINGTON' OR = 'LOS ANGELES'
AND BIRTH GT 19560101
```

```
FIND CARS WITH MAKE = 'VOLKSWAGEN'
AND COLOR = 'RED' OR = 'BLUE' OR = 'BLACK'
```

Such-Schlüsselfelder (Deskriptoren) für Adabas-Dateien

Adabas-Benutzer können bei der Suche nach Datensätzen als Suchschlüssel Datenbankfelder, die als Deskriptoren definiert sind, verwenden.

Subdeskriptoren, Superdeskriptoren, Hyperdeskriptoren und phonetische Deskriptoren

Bei Adabas-Datenbanken können für die Konstruktion von Suchkriterien Subdeskriptoren, Superdeskriptoren, Hyperdeskriptoren und phonetische Deskriptoren verwendet werden.

- Ein Subdeskriptor ist ein Schlüsselfeld, das Teil eines Feldes ist.
- Ein Superdeskriptor ist ein Schlüsselfeld, das aus mehreren Feldern/ Teilfeldern besteht.
- Ein Hyperdeskriptor ist ein Schlüsselfeld, das durch einen benutzerdefinierten Algorithmus gebildet wird.
- Ein phonetischer Deskriptor ermöglicht die Suche nach einem Feldwert anhand des Klanges eines Wertes. Bei der Suche mit einem phonetischen Deskriptor werden alle Werte gefunden, die so ähnlich klingen wie der Suchwert.

Bei welcher Datei welche Felder als Deskriptoren, Sub-, Super-, Hyper- und phonetische Deskriptoren verwendet werden können, ist in dem betreffenden DDM definiert.

Werte für Subdeskriptoren, Superdeskriptoren, phonetische Deskriptoren

Die mit Subdeskriptoren, Superdeskriptoren und phonetischen Deskriptoren angegebenen Suchwerte müssen mit dem jeweiligen internen Format kompatibel sein: ein Subdeskriptor hat dasselbe interne Format wie das Feld, von dem er ein Teil ist; ein phonetischer Deskriptor hat immer alphanumerisches Format; das interne Format eines Superdeskriptors ist binär, falls alle Felder, aus denen er sich zusammensetzt, numerisches Format haben; andernfalls ist sein internes Format alphanumerisch.

Werte für Sub- und Superdeskriptoren können wie folgt angegeben werden:

- als numerische oder hexadezimale Konstanten; hat ein Superdeskriptor binäres Format (vgl. oben), muß eine hexadezimale Konstante als Wert angegeben werden
- als Werte von Benutzervariablen, die mit einem REDEFINE-Statement redefiniert wurden, um die Teile auszuwählen, die den Sub- bzw. Superdeskriptorwert darstellen.

Deskriptoren aus Datenbank-Arrays

Ein Deskriptor, der Teil eines Datenbank-Arrays ist, kann ebenfalls als Suchfeld verwendet werden. Bei Adabas-Dateien kann dies ein multiples Feld sein oder ein Feld, das in einer Periodengruppe enthalten ist.

Ein Deskriptorfeld, das Teil einer Periodengruppe ist, kann entweder mit oder ohne Index angegeben werden. Wird kein Index angegeben, so wird ein Datensatz ausgewählt, wenn der Suchwert in irgendeiner Ausprägung gefunden wird. Wird ein Index angegeben, so wird ein Datensatz nur ausgewählt, wenn der Suchwert in der im Index angegebenen Ausprägung gefunden wird. Der Index muß als Konstante angegeben werden. Es darf kein Indexbereich angegeben werden.

Ist das angegebene Deskriptorfeld ein multiples Feld, darf kein Index angegeben werden. Ein Datensatz wird ausgewählt unabhängig davon, in welcher Ausprägung des Feldes der Suchwert gefunden wird.

Beispiele mit Datenbank-Arrays:

Die folgenden Beispiele basieren auf der Annahme, daß die Felder SALARY und LANG Deskriptoren sind, wobei SALARY Teil einer Periodengruppe ist und LANG ein multiples Feld.

1. FIND EMPLOYEES WITH SALARY LT 20000

(sucht alle Ausprägungen von SALARY ab)

2. FIND EMPLOYEES WITH SALARY (1) LT 20000

(sucht nur in der ersten Ausprägung von SALARY)

3. FIND EMPLOYEES WITH SALARY (1:4) LT 20000 /* ungültig

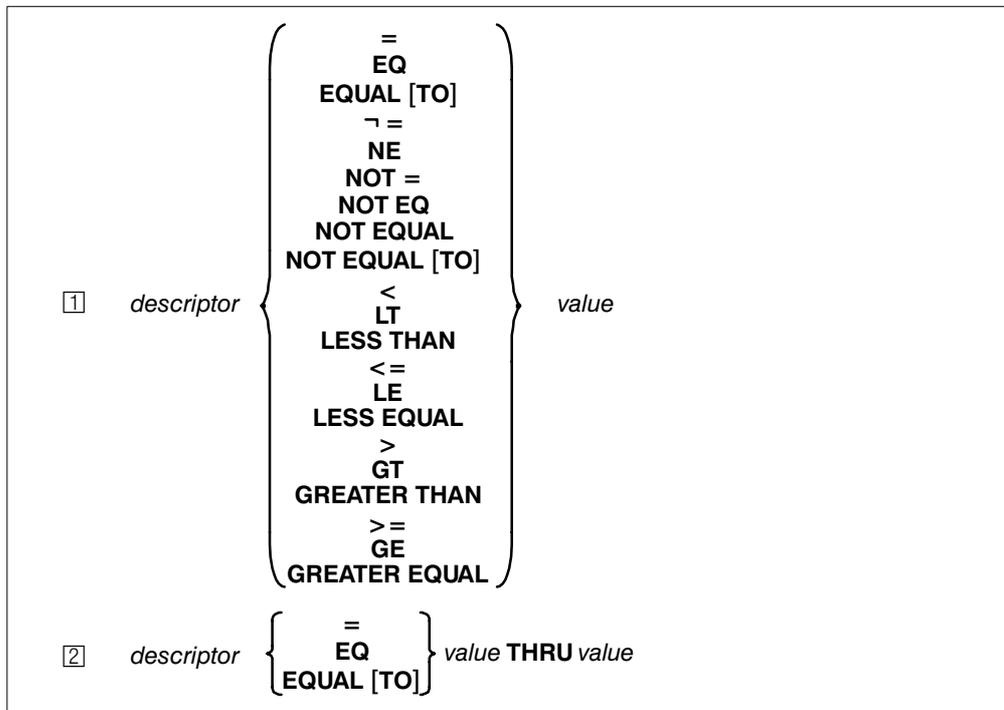
(ein Bereich von Ausprägungen darf nicht angegeben werden, wenn ein Feld aus einer Periodengruppe als Suchkriterium verwendet wird)

4. FIND EMPLOYEES WITH LANG = 'FRENCH'

(sucht alle Werte von LANG ab)

5. FIND EMPLOYEES WITH LANG (1) = 'FRENCH' /*ungültig

(wird ein multiples Feld als Suchkriterium verwendet, darf kein Index angegeben werden)

Suchkriterium für VSAM-Dateien (*basic-search-criterion*)

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Descriptor	S A	A N P B	nein	nein
Value	C S	A N P B	ja	nein

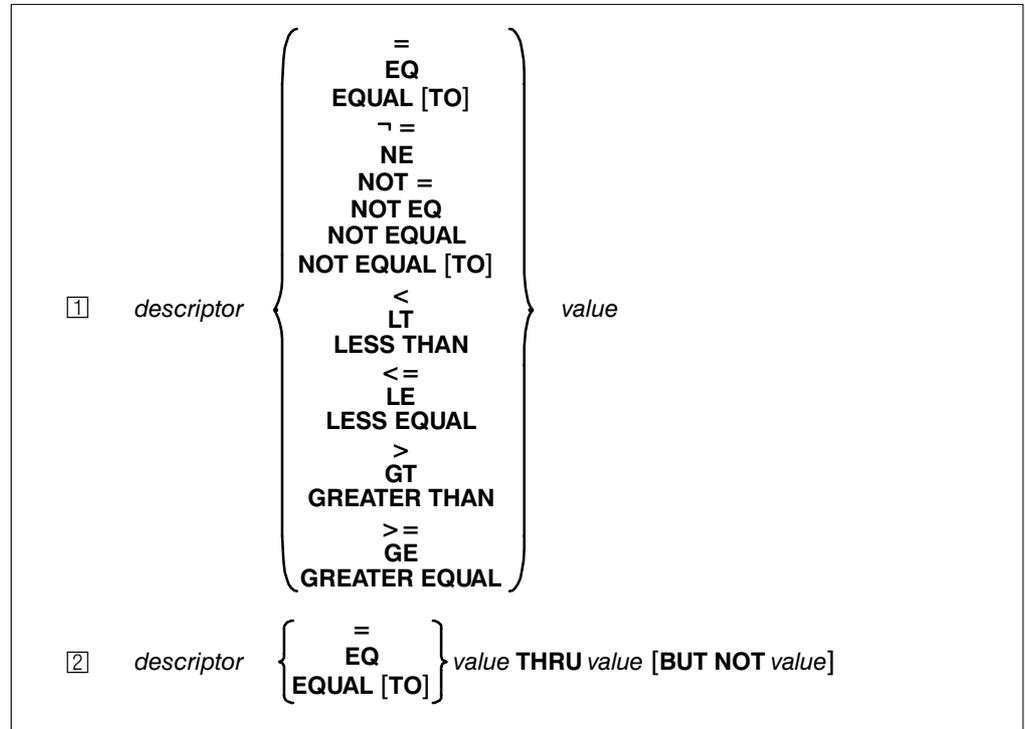
descriptor

Es kann ein Deskriptor angegeben werden, der in einer VSAM-Datei als VSAM-Schlüsselfeld definiert und im entsprechenden Natural-DDM mit “P” (Primärschlüssel) oder “A” (Alternativschlüssel) markiert ist.

value

Der Suchwert, den das Schlüsselfeld haben soll.

Die Formate von *descriptor* und Suchwert (*value*) müssen kompatibel sein.

Suchkriterium für DL/I-Dateien (*basic-search-criterion*)

Operand	Mögliche Struktur	Mögliche Formate		Referenzierung erlaubt	Dynam. Definition
Descriptor	S A	A N P	B	nein	nein
Value	C S	A N P	B	ja	nein

descriptor

Es kann ein Deskriptor angegeben werden, der in DL/I als Feld definiert und im entsprechenden Natural-DDM mit “D” markiert ist.

value

Der Suchwert, den das Deskriptorfeld haben soll.

Für HDAM-Datenbanken ist nur das folgende *basic-search-criterion* möglich:

$$\text{descriptor} \left\{ \begin{array}{c} = \\ \text{EQ} \\ \text{EQUAL [TO]} \end{array} \right\} \text{value}$$

Verknüpfen von Suchkriterien (für DL/I-Dateien)

$$[\text{NOT}] \left\{ \begin{array}{l} \textit{basic-search-criterion} \\ \textit{(search-expression)} \end{array} \right\} \left[\left\{ \begin{array}{l} \text{OR} \\ \text{AND} \end{array} \right\} \textit{search-expression} \right] \dots$$

basic-search-criteria, die sich auf unterschiedliche Segment-Arten beziehen, dürfen mit dem logischen Operator “OR” nicht miteinander verknüpft werden.

Beispiele:

```
FIND COURSE WITH COURSEN > 1
```

```
FIND COURSE WITH COURSEN > 1 AND COURSEN < 100
```

```
FIND OFFERING WITH (COURSEN-COURSE > 1 OR TITLE-COURSE = 'NATURAL')
                    AND LOCATION = 'DARMSTADT'
```

Ungültige Verknüpfung:

```
FIND OFFERING WITH COURSEN-COURSE > 1 OR LOCATION = 'DARMSTADT'
```

COUPLED-clause

Diese Klausel gilt nur für Zugriffe auf Adabas-Dateien.

Mit Entire System Server darf diese Klausel nicht verwendet werden.

$\left\{ \begin{array}{l} \text{AND} \\ \text{OR} \end{array} \right\} \text{ COUPLED } [\text{TO}] \text{ } [\text{FILE}] \text{ view-name}$ $\left[\text{VIA descriptor1} \left\{ \begin{array}{l} = \\ \text{EQ} \\ \text{EQUAL } [\text{TO}] \end{array} \right\} \text{ descriptor2} \right]$ $[\text{WITH}] \text{ basic-search-criteria}$

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Descriptor1	S A	A N P B	nein	nein
Descriptor2	S A	A N P B	nein	nein

Anmerkung:

Ohne VIA-Klausel kann die COUPLED-Klausel bis zu viermal angegeben werden, mit VIA-Klausel bis zu 42-mal.

Adabas bietet die Möglichkeit, Dateien miteinander zu koppeln. Dadurch ist es mit der COUPLED-Klausel möglich, im Suchkriterium eines einzigen FIND-Statements Deskriptoren von verschiedenen Dateien anzugeben.

Zwei verschiedene COUPLED-Klauseln desselben FIND-Statements dürfen nicht dieselbe Adabas-Datei verwenden.

Ein *set-name* (siehe *RETAIN-Klausel*) darf nicht im Suchkriterium (*basic-search-criteria*) angegeben werden.

Datenbankfelder der in der COUPLED-Klausel angegebenen Datei können anschließend im Programm nicht referenziert werden, wenn auf diese Datei kein separater FIND- oder READ-Zugriff erfolgt.

Anmerkung:

Wenn die COUPLED-Klausel verwendet wird, kann die Haupt-WITH-Klausel gegebenenfalls weggelassen werden. Wenn die Haupt-WITH-Klausel weggelassen wird, dürfen die Schlüsselwörter AND/OR in der COUPLED-Klausel nicht angegeben werden.

Physisches Koppeln (ohne VIA-Klausel)

Die in der COUPLED-Klausel ohne VIA verwendeten Dateien müssen mit der entsprechenden Adabas-Utility physisch gekoppelte Adabas-Dateien sein (wie in der Adabas-Dokumentation beschrieben).

Beispiel mit physisch gekoppelten Dateien:

```

/* EXAMPLE 'FNDCPL': FIND (USING COUPLED FILES)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 MAKE
END-DEFINE
/*****
FIND EMPLOY-VIEW WITH CITY = 'FRANKFURT'
      AND COUPLED TO VEHIC-VIEW WITH MAKE = 'VW'
  DISPLAY NOTITLE NAME
END-FIND
/*****
END

```

Die Referenzierung von NAME im DISPLAY-Statement ist gültig, da dieses Feld in der Datei EMPLOYEES enthalten ist; eine Referenzierung von MAKE hingegen wäre nicht gültig, da MAKE in der Datei VEHICLES enthalten ist, welche in der COUPLED-Klausel angegeben wurde.

In diesem Beispiel werden Datensätze nur gefunden, wenn die beiden Dateien EMPLOYEES und VEHICLES physisch gekoppelt sind.

Logisches Koppeln (VIA-Klausel)

Die Option “VIA *descriptor1* = *descriptor2*” erlaubt es Ihnen, in einer Suchabfrage mehrere Adabas-Dateien logisch miteinander zu koppeln. *Descriptor1* ist ein Feld aus dem ersten View, *descriptor2* ein Feld aus dem zweiten View. Die beiden Dateien brauchen nicht physisch in Adabas gekoppelt zu sein. Diese COUPLED-Option nutzt die mit Adabas Version 5 gebotene Möglichkeit des “Soft Coupling” aus, die in der Adabas-Dokumentation beschrieben ist.

Beispiel mit VIA-Klausel:

```

/* EXAMPLE 'FNSEX1': FIND (USING SOFT COUPLING)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
END-DEFINE
/*****
FIND EMPLOY-VIEW WITH NAME = 'ADKINSON'
  AND COUPLED TO VEHIC-VIEW
    VIA PERSONNEL-ID = PERSONNEL-ID
    WITH MAKE = 'VOLVO'
  DISPLAY PERSONNEL-ID NAME FIRST-NAME
END-FIND
/*****
END

```

Page	1	91-06-18	14:30:38
PERSONNEL ID	NAME	FIRST-NAME	
20011000	ADKINSON	BOB	

STARTING WITH ISN=*operand5*

Diese Klausel gilt nur für Zugriffe auf Adabas- und VSAM-Datenbanken, und zwar bei VSAM nur für ESDS.

Mit dieser Klausel können Sie als Startwert (*operand5*) eine Adabas-ISN (Interne Satznummer) bzw. VSAM-RBA (relative Byte-Adresse) angeben, ab der Datensätze gelesen werden sollen.

Diese Klausel kann zum Repositionieren innerhalb einer FIND-Schleife, deren Verarbeitung unterbrochen wurde, benutzt werden, um auf einfache Weise den nächsten Datensatz zu bestimmen, mit dem die Verarbeitung fortgesetzt werden soll. Dies ist besonders hilfreich, wenn der nächste Datensatz sich nicht eindeutig durch einen seiner Deskriptorwerte ermitteln läßt.

Es kann auch hilfreich sein in einer verteilten Client/Server-Anwendung, in der das Lesen der Datensätze von einem Server-Programm und die weitere Verarbeitung der Datensätze von einem Client-Programm durchgeführt werden, wobei die Datensätze nicht alle auf einmal, sondern schubweise verarbeitet werden.

Anmerkung:

*Als tatsächlicher Startwert wird nicht der Wert von *operand5*, sondern der nächsthöhere Wert genommen.*

Beispiel:

Siehe Programm FNDSISN in Library SYSEXRM.

SORTED BY-clause

*Diese Klausel gilt nur für Zugriffe auf Adabas- und SQL-Datenbanken.
Mit Entire System Server darf diese Klausel nicht verwendet werden.*

SORTED [BY] descriptor...3 [DESCENDING]

Die SORTED BY-Klausel dient dazu, die ausgewählten Datensätze in der Reihenfolge der Werte eines oder mehrerer (maximal 3) Deskriptoren zu sortieren. Diese Deskriptoren brauchen nicht mit den als Suchkriterium verwendeten Deskriptoren identisch sein.

Normalerweise wird in *aufsteigender* Reihenfolge der Werte sortiert; wünschen Sie eine *absteigende* Sortierfolge, geben Sie das Schlüsselwort DESCENDING an. Der Sortiervorgang verwendet die Adabas-Invertierten-Listen, es werden hierbei keine Datensätze gelesen.

Anmerkung:

Diese Klausel kann beträchtliche zusätzliche Verarbeitungszeit beanspruchen, falls das zur Steuerung der Sortierfolge verwendete Deskriptorfeld viele Werte enthält. Das liegt daran, daß die gesamte Wertliste abgesucht werden muß, bis alle ausgewählten Datensätze lokalisiert worden sind. Wollen Sie eine große Zahl von Datensätzen sortieren, sollten Sie daher stattdessen das Statement SORT verwenden.

Bei der Verwendung einer SORTED BY-Klausel gelten die Adabas-Sortierlimits (siehe ADARUN-Parameter LS in der Adabas-Dokumentation).

In einer SORTED BY-Klausel darf kein Deskriptor, der Teil einer Periodengruppe ist, angegeben werden; ein multiples Feld (ohne Index) darf angegeben werden.

Außer auf OpenVMS und Großrechnern dürfen in einer SORTED BY-Klausel auch Nicht-Deskriptoren angegeben werden.

Eine *SORTED BY-Klausel* darf nicht zusammen mit einer *RETAIN-Klausel* verwendet werden.

Beispiel für SORTED BY-Klausel:

```

/* EXAMPLE 'FNDSOR': FIND (SORTED BY CLAUSE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 FIRST-NAME
  2 PERSONNEL-ID
END-DEFINE
/*****
LIMIT 10
FIND EMPLOY-VIEW WITH CITY = 'FRANKFURT'
      SORTED BY NAME PERSONNEL-ID
  DISPLAY NOTITLE NAME (IS=ON) FIRST-NAME PERSONNEL-ID
END-FIND
/*****
END

```

NAME	FIRST-NAME	PERSONNEL ID
BAECKER	JOHANNES	11500345
BECKER	HERMANN	11100311
BERGMANN	HANS	11100301
BLAU	SARAH	11100305
BLOEMER	JOHANNES	11200312
DIEDRICHS	HUBERT	11600301
DOLLINGER	MARGA	11500322
FALTER	CLAUDIA	11300311
	HEIDE	11400311
FREI	REINHILD	11500301

RETAIN-clause

Diese Klausel gilt nur für Zugriffe auf Adabas-Datenbanken.
Mit Entire System Server darf diese Klausel nicht verwendet werden.

RETAIN AS operand6				
Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand6	C S	A	ja	nein

Mit der RETAIN-Klausel ist es möglich, das Ergebnis einer ausgedehnten Suche in einer großen Datei für die weitere Verarbeitung zurückzustellen. Die ausgewählten Datensätze werden als "ISN-Set" in die Adabas-Arbeitsdatei gestellt. Dieser Set kann in nachfolgenden FIND-Statements als Suchkriterium (*basic-search-criterion*) angegeben werden, um aus dem Set eine gezieltere Auswahl von Datensätzen für die weitere Verarbeitung zu treffen. Der Set ist dateispezifisch und kann nur von einem anderen FIND-Statement verwendet werden, das dieselbe Datei verarbeitet. Der Set kann von einem beliebigen anderen Natural-Programm referenziert werden.

Set-Name (*operand6*)

Der Set-Name identifiziert den "ISN-Set". Der Name kann als alphanumerische Konstante oder in Form einer alphanumerischen Benutzervariablen angegeben werden. Es wird nicht geprüft, ob ein Set-Name bereits vergeben ist; wird ein Set-Name zweimal vergeben, überschreibt der neue Set den alten.

Freigabe von Sets

Die Anzahl der Sets, die zurückgestellt werden können, ist nicht begrenzt; die Anzahl der ISNs pro Set auch nicht. Die zu einer gegebenen Zeit benötigte Mindestanzahl von ISN-Sets sollte definiert werden. Nicht länger benötigte Sets sollten mit einem RELEASE SETS-Statement aus der Arbeitsdatei gelöscht werden.

Erstellte ISN-Sets bleiben während der gesamten Natural-Session erhalten und werden nicht automatisch gelöscht, außer bei einem Library-Wechsel. Ein mit einem Programm erstellter Set kann von einem anderen Programm referenziert und verarbeitet oder unter Angabe zusätzlicher Suchkriterien weiter selektiert werden.

Zugriffe anderer Benutzer

Die Datensätze, deren ISNs in einem ISN-Set enthalten sind, sind nicht vor Zugriff/Veränderung durch andere Benutzer geschützt. Bevor Sie Datensätze aus dem Set verarbeiten, empfiehlt es sich daher sicherzustellen, daß das ursprüngliche Selektionskriterium, mit dem der Set erstellt wurde, immer noch auf die ausgewählten Datensätze zutrifft.

Dies geschieht mit einem neuen FIND-Statement, bei dem man den Set-Namen in der WITH-Klausel als primäres Suchkriterium angibt und in einer WHERE-Klausel das ursprüngliche Primärsuchkriterium (d.h. das Suchkriterium aus der WITH-Klausel des FIND-Statements, mittels dessen der Set erstellt wurde).

Einschränkung

Eine *RETAIN-Klausel* darf nicht zusammen mit einer *SORTED BY-Klausel* verwendet werden.

Beispiel für RETAIN-Klausel:

```

* EXAMPLE 'RELEX1': FIND (RETAIN CLAUSE) AND RELEASE
*****
DEFINE DATA LOCAL
  1 EMPLOY-VIEW VIEW OF EMPLOYEES
    2 CITY
    2 BIRTH
    2 NAME
  1 #BIRTH (D)
END-DEFINE
*
MOVE EDITED '19400101' TO #BIRTH (EM=YYYYMMDD)
*
FIND NUMBER EMPLOY-VIEW WITH BIRTH GT #BIRTH
      RETAIN AS 'AGESET1'
IF *NUMBER = 0
  STOP
END-IF
*
FIND EMPLOY-VIEW WITH 'AGESET1' AND CITY = 'NEW YORK'
  DISPLAY NOTITLE NAME CITY BIRTH (EM=YYYY-MM-DD)
END-FIND
*
RELEASE SET 'AGESET1'
END

```

NAME	CITY	DATE OF BIRTH
RUBIN	NEW YORK	1945-10-27
WALLACE	NEW YORK	1945-08-04

WHERE-clause

WHERE *logical-condition*

Die WHERE-Klausel dient dazu, ein zusätzliches Selektionskriterium anzugeben, das ausgewertet wird, *nachdem* ein über die WITH-Klausel ausgewählter Datensatz gelesen wurde und *bevor* ein ausgewählter Datensatz weiter verarbeitet wird (einschließlich AT BREAK-Auswertung).

Die für logische Bedingungen gültige Syntax ist im Abschnitt **Logische Bedingungen** im *Natural Referenzhandbuch* erklärt.

Ist für das FIND-Statement die Anzahl der zu verarbeitenden Datensätze durch ein Limit begrenzt, so werden Datensätze, die aufgrund einer WHERE-Klausel nicht weiterverarbeitet werden, bei der Ermittlung des Limits *nicht* mitgezählt. Sie werden allerdings bei der Ermittlung eines auf allgemeinerer Ebene gesetzten Limits (Session-Parameter, GLOBALS-Kommando oder LIMIT-Statement) mitgezählt.

Beispiel für WHERE-Klausel:

```

/* EXAMPLE 'FNDWHE': FIND (WHERE CLAUSE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 JOB-TITLE
  2 CITY
END-DEFINE
/*****
FIND EMPLOY-VIEW WITH CITY = 'PARIS'
      WHERE JOB-TITLE = 'INGENIEUR COMMERCIAL'
  DISPLAY NOTITLE CITY JOB-TITLE PERSONNEL-ID NAME
END-FIND
/*****
END

```

CITY	CURRENT POSITION	PERSONNEL ID	NAME
PARIS	INGENIEUR COMMERCIAL	50007300	CAHN
PARIS	INGENIEUR COMMERCIAL	50006500	MAZUY
PARIS	INGENIEUR COMMERCIAL	50004400	VALLY
PARIS	INGENIEUR COMMERCIAL	50002800	BRETON
PARIS	INGENIEUR COMMERCIAL	50001000	GIGLEUX

*IF NO RECORDS FOUND-clause***Structured-Mode-Syntax**

```

IF NO [RECORDS] [FOUND]
  { ENTER
    statement... }
END—NOREC

```

Reporting-Mode-Syntax

```

IF NO [RECORDS] [FOUND]
  { ENTER
    statement
    DO statement... DOEND }

```

In der IF NO RECORDS FOUND-Klausel können Sie eine Verarbeitung angeben, die ausgeführt werden soll für den Fall, daß kein Datensatz die in der WITH- und WHERE-Klausel des FIND-Statements angegebenen Selektionskriterien erfüllt.

Ist dies der Fall, dann löst die IF NO RECORDS FOUND-Klausel eine Verarbeitungsschleife aus, die einmal mit einem "leeren" Datensatz durchlaufen wird. Falls Sie dies nicht wünschen, geben Sie in der IF NO RECORDS FOUND-Klausel das Statement ESCAPE BOTTOM an.

Enthält die IF NO RECORDS FOUND-Klausel ein oder mehrere Statements, werden diese ausgeführt, unmittelbar bevor die Schleife durchlaufen wird. Sollen vor Durchlaufen der Schleife keine weiteren Statements ausgeführt werden, muß die IF NO RECORDS FOUND-Klausel das Schlüsselwort ENTER enthalten.

Datenbankwerte

Natural setzt alle Datenbankfelder, die die in der aktuellen Verarbeitungsschleife angegebene Datei referenzieren, auf Leerwerte, es sei denn, eines der in der IF NO RECORDS FOUND-Klausel angegebenen Statements weist den Feldern andere Werte zu.

Auswertung von Systemfunktionen

Natural-Systemfunktionen werden einmal für den leeren Datensatz ausgewertet, der für die aus der IF NO RECORDS FOUND-Klausel resultierende Verarbeitung erstellt wurde.

Einschränkung

Bei FIND FIRST, FIND NUMBER und FIND UNIQUE kann diese Klausel nicht verwendet werden.

Beispiel für IF NO RECORDS FOUND-Klausel:

```

/* EXAMPLE 'FNDIFN': FIND (IF NO RECORDS FOUND CLAUSE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
/*****
LIMIT 15
EMP. READ EMPLOY-VIEW BY NAME STARTING FROM 'JONES'
/*****
VEH. FIND VEHIC-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (EMP.)
  IF NO RECORDS FOUND
    MOVE '*** NO CAR ***' TO MAKE
  END-NOREC
  DISPLAY NOTITLE
    NAME (EMP.) (IS=ON) FIRST-NAME (EMP.) (IS=ON) MAKE (VEH.)
  END-FIND
/*****
END-READ
END

```

NAME	FIRST-NAME	MAKE
JONES	VIRGINIA	CHRYSLER
	MARSHA	CHRYSLER
		CHRYSLER
	ROBERT	GENERAL MOTORS
	LILLY	FORD
		MG
	EDWARD	GENERAL MOTORS
	MARTHA	GENERAL MOTORS
	LAUREL	GENERAL MOTORS
	KEVIN	DATSUN
JOPER	GREGORY	FORD
JOUSSELIN	MANFRED	*** NO CAR ***
JUBE	DANIEL	RENAULT
JUNG	GABRIEL	*** NO CAR ***
JUNKIN	ERNST	*** NO CAR ***
KAISER	JEREMY	*** NO CAR ***
	REINER	*** NO CAR ***

Systemvariablen im FIND-Statement

Die Natural-Systemvariablen *ISN, *NUMBER und *COUNTER werden automatisch für jedes FIND-Statement erzeugt. Wird eine Systemvariable außerhalb der aktuellen Verarbeitungsschleife oder über ein FIND FIRST-, FIND NUMBER- oder FIND UNIQUE-Statement referenziert, muß mittels Statement-Label oder Sourcecode-Zeilenummer referenziert werden. Alle drei Systemvariablen haben Format/Länge P10; diese(s) Format/Länge kann nicht geändert werden.

*ISN

Bei Adabas-Datenbanken enthält *ISN die Adabas-ISN (Interne Satz-Nummer) des gerade verarbeiteten Datensatzes. *ISN kann nicht bei FIND NUMBER verwendet werden.

Bei VSAM-Datenbanken enthält *ISN die relative Byte-Adresse (RBA) des gerade verarbeiteten Datensatzes (nur für ESDS-Dateien).

Bei DL/I- und SQL-Datenbanken und mit Entire System Server kann *ISN nicht verwendet werden.

*NUMBER

Enthält die Anzahl der Datensätze, die das in der WITH-Klausel angegebene primäre Suchkriterium erfüllt haben.

Bei DL/I-Datenbanken enthält *NUMBER den Wert "0", falls keine Segmentausprägung das Suchkriterium erfüllt, und den Wert "9999", falls mindestens eine Segmentausprägung das Suchkriterium erfüllt.

Bei VSAM-Datenbanken enthält *NUMBER nur ein sinnvolles Ergebnis, wenn im Suchkriterium der Operator EQUAL TO verwendet wird. Bei jedem anderen Operator enthält *NUMBER den Wert "0", wenn keine Datensätze gefunden wurden; jeder andere Wert bedeutet, daß Datensätze gefunden wurden, aber der Wert steht in keinem Zusammenhang mit der tatsächlichen Anzahl der gefundenen Datensätze.

Bezüglich SQL-Datenbanken siehe Kapitel **Systemvariablen** im *Natural Referenzhandbuch*.

Mit Entire System Server kann *NUMBER nicht verwendet werden.

*COUNTER

Enthält die Anzahl, wie oft die Verarbeitungsschleife durchlaufen worden ist.

Beispiel mit Systemvariablen:

```

/* EXAMPLE 'FNDVAR': FIND (SYSTEM VARIABLES *ISN, *NUMBER, *COUNTER)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 CITY
END-DEFINE
/*****
LIMIT 3
FIND EMPLOY-VIEW WITH CITY = 'MADRID'
  DISPLAY NOTITLE PERSONNEL-ID NAME
    *ISN *NUMBER *COUNTER
END-FIND
/*****
END

```

PERSONNEL ID	NAME	ISN	NMBR	CNT
60000114	DE JUAN	401	41	1
60000136	DE LA MADRID	402	41	2
60000209	PINERO	406	41	3

Mehrere FIND-Statements

Es ist möglich, mehrere FIND-Schleifen ineinander zu verschachteln. Hierbei wird eine jeweils innere Schleife für jeden Datensatz, der mit der jeweils äußeren Schleife ausgewählt wurde, durchlaufen.

Beispiel für mehrere FIND-Statements:

Im folgenden Beispiel werden zunächst alle Personen mit Namen "Smith" von der Datei EMPLOYEES ausgewählt. Dann wird die Personalnummer (PERSONNEL-ID) von der EMPLOYEES-Datei als Suchkriterium für einen Zugriff auf die Datei VEHICLES benutzt. Der erzeugte Report zeigt die (von der EMPLOYEES-Datei gelesenen) Namen und Vornamen aller Personen, die "Smith" heißen, sowie die (von der VEHICLES-Datei gelesenen) Automarken der im Besitz dieser Personen befindlichen Fahrzeuge:

```

/* EXAMPLE 'FNDMUL': FIND (USING MULTIPLE FILES)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
/*****
LIMIT 15
EMP. FIND EMPLOY-VIEW WITH NAME = 'SMITH'
VEH. FIND VEHIC-VIEW WITH PERSONNEL-ID = EMP.PERSONNEL-ID
  IF NO RECORDS FOUND
    MOVE '*** NO CAR ***' TO MAKE
  END-NOREC
  DISPLAY NOTITLE
    EMP.NAME (IS=ON)
    EMP.FIRST-NAME (IS=ON) VEH.MAKE
  END-FIND
END-FIND
/*****
END

```

NAME	FIRST-NAME	MAKE
SMITH	GERHARD	ROVER
	SEYMOUR	*** NO CAR ***
	MATILDA	FORD
	ANN	*** NO CAR ***
	TONI	TOYOTA
	MARTIN	*** NO CAR ***
	THOMAS	FORD
	SUNNY	*** NO CAR ***
	JUNE	JAGUAR
	MARK	FORD
	LOUISE	CHRYSLER
	MAXWELL	MERCEDES-BENZ
		MERCEDES-BENZ
	ELSA	CHRYSLER
	CHARLY	CHRYSLER
LEE	*** NO CAR ***	

FIND FIRST

Das FIND FIRST-Statement dient dazu, den ersten Datensatz, der die WITH- und WHERE-Selektionskriterien erfüllt, auszuwählen und zu verarbeiten.

Bei Adabas-Dateien wird derjenige der ausgewählten Datensätze verarbeitet, der die niedrigste Adabas-ISN (Interne Satznummer) hat.

FIND FIRST initiiert *keine* Verarbeitungsschleife.

Einschränkungen

FIND FIRST darf nur im Reporting Mode verwendet werden.

FIND FIRST ist beim Zugriff auf DL/I- und SQL-Datenbanken nicht möglich.

Ein FIND FIRST-Statement darf keine IF NO RECORDS FOUND-Klausel enthalten.

Systemvariablen mit FIND FIRST

Mit dem FIND FIRST-Statement stehen folgende Natural-Systemvariablen zur Verfügung:

*ISN

Enthält die Adabas-ISN (Interne Satznummer) des ausgewählten Datensatzes. *ISN enthält den Wert "0", wenn kein Datensatz die WITH- und WHERE-Selektionskriterien erfüllt.

*ISN steht für VSAM-Datenbanken nicht zur Verfügung. Mit Entire System Server kann *ISN nicht verwendet werden.

*NUMBER

Enthält die Anzahl der Datensätze, die vor Auswertung des WHERE-Kriteriums das in der WITH-Klausel angegebene primäre Selektionskriterium erfüllt haben. *NUMBER enthält den Wert "0", wenn kein Datensatz das WITH-Kriterium erfüllt.

Mit Entire System Server kann *NUMBER nicht verwendet werden.

*COUNTER

Enthält "1", wenn ein Datensatz gefunden wurde; enthält "0", wenn kein Datensatz gefunden wurde.

Beispiel für FIND FIRST

Siehe Programm FNDFIR in Library SYSEXRM.

FIND NUMBER

Mit dem Statement FIND NUMBER können Sie ermitteln, wieviele Datensätze die WITH- und WHERE-Kriterien erfüllen. FIND NUMBER löst keine Verarbeitungsschleife aus und *liest auch keine Datensätze von der Datenbank.*

Anmerkung:

Eine WHERE-Klausel kann hierbei einen beträchtlichen Verarbeitungsmehraufwand verursachen.

Einschränkungen

Ein FIND NUMBER-Statement darf keine SORTED BY- oder IF NO RECORDS FOUND-Klausel enthalten sowie im *Structured Mode* auch keine WHERE-Klausel.

Die WHERE-Klausel kann im *Structured Mode* nicht verwendet werden.

FIND NUMBER ist beim Zugriff auf DL/I-Datenbanken nicht möglich.

Mit Entire System Server kann FIND NUMBER nicht verwendet werden.

Systemvariablen mit FIND NUMBER

Mit FIND NUMBER stehen folgende Natural-Systemvariablen zur Verfügung:

*NUMBER

Enthält die Anzahl der Datensätze, die das in der WITH-Klausel angegebene primäre Selektionskriterium erfüllt haben.

*COUNTER

Enthält die Anzahl der Datensätze, die die Selektionskriterien der WHERE-Klausel erfüllt haben.

*COUNTER steht nur zur Verfügung, wenn das FIND NUMBER-Statement eine WHERE-Klausel enthält.

Beispiel für FIND NUMBER:

```
* EXAMPLE 'FNDNUM': FIND NUMBER
*****
DEFINE DATA LOCAL
  1 EMPLOY-VIEW VIEW OF EMPLOYEES
    2 CITY
    2 BIRTH
  1 #BIRTH (D)
END-DEFINE
*
MOVE EDITED '19500101' TO #BIRTH (EM=YYYYMMDD)
*
FIND NUMBER EMPLOY-VIEW WITH CITY = 'MADRID'
WHERE BIRTH LT #BIRTH
*
WRITE NOTITLE 'TOTAL RECORDS SELECTED:      ' *NUMBER
              / 'TOTAL BORN BEFORE 1 JAN 1950: ' *COUNTER
*
END
```

TOTAL RECORDS SELECTED:	41
TOTAL BORN BEFORE 1 JAN 1950:	16

FIND UNIQUE

Dieses Statement gewährleistet, daß nur ein einziger Datensatz die Selektionskriterien erfüllt. FIND UNIQUE initiiert *keine* Verarbeitungsschleife. Enthält das FIND UNIQUE-Statement eine WHERE-Klausel, wird zur Auswertung dieser Klausel eine automatische interne Verarbeitungsschleife durchlaufen.

Wird kein oder mehr als ein Datensatz gefunden, wird eine entsprechende Fehlermeldung ausgegeben; dieser Fall kann mit einem ON ERROR-Statement abgefangen werden.

Einschränkungen

FIND UNIQUE darf nur im Reporting Mode verwendet werden.

FIND UNIQUE ist beim Zugriff auf DL/I-Datenbanken sowie beim Einsatz von Entire System Server nicht möglich.

Bei SQL-Datenbanken ist FIND UNIQUE nicht erlaubt. (Ausnahme: auf Großrechnern kann FIND UNIQUE für Primärschlüssel verwendet werden; allerdings ist dies nur aus Kompatibilitätsgründen erlaubt und sollte nicht benutzt werden.)

Ein FIND UNIQUE-Statement darf keine SORTED BY- oder IF NO RECORDS FOUND-Klausel enthalten.

Beispiel für FIND UNIQUE

Siehe Programm FNDUNQ in Library SYSEXRM.

FOR

<pre> FOR operand1 [[:]= EQ FROM] operand2 [TO THRU] operand3 [[STEP] operand4] statement... END-FOR (nur Structured Mode) [LOOP] (nur Reporting Modey) </pre>

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S	N P I F	ja	ja
Operand2	C S N	N P I F	ja	nein
Operand3	C S N	N P I F	ja	nein
Operand4	C S N	N P I F	ja	nein

Verwandtes Statement

REPEAT.

Funktion

Mit dem Statement FOR wird eine Verarbeitungsschleife ausgelöst und gleichzeitig die Anzahl der Schleifendurchläufe gesteuert.

Kontrollvariable (*operand1*) und Ausgangswert (*operand2*)

Operand1 kann ein Datenbankfeld oder eine Benutzervariable sein und steuert die Anzahl der Schleifendurchläufe. Der nach dem Schlüsselwort FROM angegebene Wert (*operand2*) wird der Kontrollvariablen als Ausgangswert zugeordnet, bevor die Verarbeitungsschleife das erste Mal durchlaufen wird. Dieser Ausgangswert erhöht sich mit jedem Schleifendurchlauf um den Wert des mit STEP angegebenen *operand4* (bzw. vermindert sich, wenn *operand4* einen negativen Wert hat).

Die Kontrollvariable kann während der Ausführung der Verarbeitungsschleife referenziert werden, um den aktuellen Wert der Kontrollvariablen zu erhalten.

TO-Wert (*operand3*)

Die Verarbeitungsschleife wird beendet, sobald *operand1* einen Wert enthält, der größer ist als der Wert von *operand3* (oder kleiner, falls der STEP-Wert negativ ist).

STEP-Wert (*operand4*)

Der Wert von *operand4* kann positiv oder negativ sein. Ist kein Wert angegeben, wird ein Wert von "+1" angenommen.

Je nach dem Vorzeichen des STEP-Wertes wird die Vergleichsoperation für *operand3* auf "größer als" oder "kleiner als" gesetzt, wenn die Verarbeitungsschleife zum erstenmal durchlaufen wird.

Operand4 darf nicht "0" sein.

Konsistenzprüfung

Bevor die FOR-Schleife zum erstenmal durchlaufen wird, wird geprüft, ob die Werte der Operanden konsistent sind (d.h. ob es möglich ist, daß durch wiederholtes Addieren von *operand4* zu *operand2* der Wert von *operand3* erreicht werden kann); ist dies nicht der Fall, wird die FOR-Schleife nicht durchlaufen (ohne daß eine Fehlermeldung ausgegeben wird).

Beispiel

```

/* EXAMPLE 'FOREX1S': FOR (STRUCTURED MODE)
/*****
DEFINE DATA LOCAL
1 #INDEX (I1)
1 #ROOT (N2.7)
END-DEFINE
/*****
FOR #INDEX 1 TO 5
  COMPUTE #ROOT = SQRT (#INDEX)
  WRITE NOTITLE '=' #INDEX 3X '=' #ROOT
END-FOR
/*****
SKIP 1
FOR #INDEX 1 TO 5 STEP 2
  COMPUTE #ROOT = SQRT (#INDEX)
  WRITE '=' #INDEX 3X '=' #ROOT
END-FOR
/*****
END

```

#INDEX:	1	#ROOT:	1.0000000
#INDEX:	2	#ROOT:	1.4142135
#INDEX:	3	#ROOT:	1.7320508
#INDEX:	4	#ROOT:	2.0000000
#INDEX:	5	#ROOT:	2.2360679
#INDEX:	1	#ROOT:	1.0000000
#INDEX:	3	#ROOT:	1.7320508
#INDEX:	5	#ROOT:	2.2360679

Äquivalentes Reporting-Mode-Beispiel: siehe Programm FOREX1R in Library SYSEXRM.

FORMAT

FORMAT [(*rep*)] *parameter...*

Funktion

Das Statement **FORMAT** dient dazu, Werte für Eingabe- und Ausgabeparameter festzusetzen. Die Gültigkeit der mit einem **FORMAT**-Statement festgesetzten Werte hat (zur Kompilierungszeit) Vorrang vor den auf Session-Ebene (mit einem **GLOBALS**-Kommando, **SET GLOBALS**-Statement oder vom Natural-Administrator) gesetzten Parameterwerten. Die mit dem **FORMAT**-Statement festgesetzten Werte können ihrerseits auf Statement- oder Feldebene von den mit den Statements **DISPLAY**, **INPUT**, **PRINT**, **WRITE**, **WRITE TITLE** oder **WRITE TRAILER** gesetzten Parameterwerten überschrieben werden.

Falls die mit **FORMAT** festgelegten Werte nicht auf Statement- oder Feldebene überschrieben werden, gelten Sie bis zum Ende des betreffenden Programms oder bis sie mit einem weiteren **FORMAT**-Statement geändert werden.

Das **FORMAT**-Statement generiert keinen ausführbaren Code im Natural-Programm. Seine Ausführung hängt *nicht* vom logischen Ablauf des Programms ab. Es wird während der *Kompilierung* ausgewertet, um die Parameter für die Kompilierung der betroffenen **DISPLAY**-, **WRITE**- und **INPUT**-Statements zu setzen. Das **FORMAT**-Statement wirkt sich auf alle nachfolgenden **DISPLAY**-, **WRITE**- und **INPUT**-Statements aus. Innerhalb eines Programms können Sie mehrere **FORMAT**-Statements verwenden, aber jeweils nur eins pro Report.

Report-Spezifikation (*rep*)

Mit der Notation “(*rep*)” kann ein bestimmter anderer Report angegeben werden, auf den sich das Statement beziehen soll. Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem **DEFINE PRINTER**-Statement zugewiesen wurde, angegeben werden. Falls nichts anderes angegeben wird, bezieht sich das **FORMAT**-Statement auf den ersten Report (Report 0).

Parameter

Die Parameter können in beliebiger Reihenfolge angegeben werden und müssen jeweils durch ein oder mehr Leerzeichen voneinander getrennt werden. Der Eintrag für einen Parameter darf nicht über das Ende einer Sourcecode-Zeile hinausgehen.

Die hier gültigen feldsensitiven Parameter-Einstellungen kommen nur für Variablenfelder in Betracht, die in einem INPUT-, WRITE-, DISPLAY- oder PRINT-Statement des ausgewählten Reports verwendet werden. Sie haben aber keine Auswirkung auf in einem der erwähnten Statements verwendete Text-Konstanten.

Beispiel

```

DEFINE DATA LOCAL
1  VARI  (A4)   INIT <'1234>
END-DEFINE
FORMAT PM=I
WRITE 'Text'          VARI
WRITE 'Text' (PM=I)   VARI
END
/*      Output
/*      Produced
/*      -----
/*      Text 4321
/*      txeT 4321

```

Eine Beschreibung der Parameter, die Sie verwenden können, finden Sie im Kapitel **Session-Parameter** des *Natural Referenzhandbuchs*.

Beispiel

```

DEFINE DATA LOCAL
1  VARI  (A4)   INIT <'1234'>
END-DEFINE
*
DISPLAY NOHDR          'Text'          '='      VARI          /*      Output
DISPLAY NOHDR (PM=I)   'Text'          '='      VARI          /*      Produced
DISPLAY NOHDR          'Text' (PM=I)   '='      VARI (PM=I)   /*      -----
DISPLAY NOHDR          'Text' (PM=I)   '='      VARI          /*      Text 1234
DISPLAY NOHDR          'Text' (PM=I)   '='      VARI          /*      Text 4321
DISPLAY NOHDR          'Text' (PM=I)   '='      VARI          /*      txeT 4321
END                                                             /*      txeT 1234

```

Beispiel

```

/* EXAMPLE 'FMTEX1:' FORMAT
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 POST-CODE
  2 COUNTRY
END-DEFINE
/*****
FORMAT AL=7 /* ALPHANUMERIC FIELD OUTPUT LENGTH
FC=+ /* FILLER CHARACTER FOR FIELD HEADER
GC=* /* FILLER CHARACTER FOR GROUP HEADER
HC=L /* HEADER LEFT JUSTIFIED
IC=<< /* INSERT CHARACTERS
IS=ON /* IDENTICAL SUPPRESS ON
TC=>> /* TRAILING CHARACTERS
UC== /* UNDERLINE CHARACTER
ZP=OFF /* ZERO PRINT OFF
/*****
LIMIT 5
READ EMPLOY-VIEW BY NAME
  DISPLAY NOTITLE NAME 3X CITY 3X POST-CODE 3X COUNTRY
END-READ
/*****
END

```

NAME++++++	CITY++++++	POSTAL+++++	COUNTRY++++
=====	=====	=====	=====
<<ABELLAN>>	<<MADRID >>	<<28014 >>	<<E >>
<<ACHIESO>>	<<DERBY >>	<<DE3 4TR>>	<<UK >>
<<ADAM >>	<<JOIGNY >>	<<89300 >>	<<F >>
<<ADKINSO>>	<<BEDFORD>>	<<1730 >>	<<USA>>
	<<FRAMING>>	<<1701 >>	

GET

```

GET [IN] [FILE] view-name
    [PASSWORD=operand1]
    [CIPHER=operand2]
    [RECORD] { operand3 } operand4...
              *ISN [(r)]

```

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	ja	nein
Operand2	C S	N	nein	nein
Operand3	C S N	N P I B	ja	nein
Operand4	S A	A N P I F B D T L	ja	ja

Funktion

Das Statement GET dient dazu, einen Datensatz mit einer bestimmten Adabas-ISBN (Interne Satz-Nummer) zu lesen.

Das GET-Statement löst keine Verarbeitungsschleife aus.

Einschränkungen

Das GET-Statement kann nicht für DL/I- oder SQL-Datenbanken verwendet werden.

Mit Entire System Server kann das GET-Statement nicht verwendet werden.

view-name

Der Name eines Views, der entweder in einem DEFINE DATA-Statement oder in einer separaten Global oder Local Data Area definiert ist. Im *Reporting Mode* darf *view-name* auch der Name eines DDMs sein.

PASSWORD und CIPHER

Diese beiden Klauseln sind nur auf Adabas- und VSAM-Datenbanken anwendbar.

Die PASSWORD-Klausel dient dazu, ein Paßwort anzugeben, um auf Daten einer paßwort-geschützten Adabas-Datei zugreifen zu können.

Die CIPHER-Klausel dient dazu, einen Cipher-Code (Chiffrierschlüssel) anzugeben, um in chiffrierter Form gespeicherte Daten in entschlüsselter Form zu erhalten.

Näheres hierzu siehe FIND- und PASSW-Statement.

***ISN / operand3**

Die ISN kann entweder als numerische Konstante oder Benutzervariable (*operand3*) oder über die Natural-Systemvariable ***ISN** angegeben werden.

Anmerkung für VSAM-Datenbanken:

Für VSAM-ESDS muß die RBA in einer numerischen Benutzervariablen enthalten sein oder als Ganzzahl-Konstante angegeben werden. Dasselbe gilt für VSAM-RRDS, außer daß hier statt der RBA die RRN angegeben werden muß.

Referenzierung von Datenbankfeldern (*operand4*)

Im *Reporting Mode* gilt: Eine spätere Referenzierung von Datenbankfeldern, die mit einem GET-Statement gelesen wurden, muß entweder unter Angabe eines Statement-Labels oder über die Sourcecode-Zeilenummer des GET-Statements erfolgen.

Im *Structured Mode* darf *operand4* nicht angegeben werden.

Beispiel

```

/* EXAMPLE 'GETEX1': GET
/*****
DEFINE DATA LOCAL
1 PERSONS VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
1 SALARY-INFO VIEW OF EMPLOYEES
  2 NAME
  2 CURR-CODE (1:1)
  2 SALARY (1:1)
1 #ISN-ARRAY (B4/1:10)
1 #LINE-NR ( N2)
END-DEFINE
/*****
FORMAT PS=16
LIMIT 10
READ PERSONS BY NAME
  MOVE *COUNTER TO #LINE-NR
  MOVE *ISN TO #ISN-ARRAY (#LINE-NR)
  DISPLAY #LINE-NR PERSONNEL-ID NAME FIRST-NAME
/*****
AT END OF PAGE
  INPUT / 'PLEASE SELECT LINE-NR FOR SALARY INFORMATION:' #LINE-NR
  IF #LINE-NR = 1 THRU 10
    GET SALARY-INFO #ISN-ARRAY (#LINE-NR)
    WRITE / SALARY-INFO.NAME
          SALARY-INFO.SALARY (1)
          SALARY-INFO.CURR-CODE (1)
  END-IF
END-ENDPAGE
/*****
END-READ
END

```

PAGE 1 97-09-29 14:53:1

#LINE-NR	PERSONNEL ID	NAME	FIRST-NAME
1	60008339	ABELLAN	KEPA
2	30000231	ACHIESON	ROBERT
3	50005800	ADAM	EDWIN
4	20005700	ADKINSON	TIMMIE
5	20008600	ADKINSON	MARTHA
6	20008800	ADKINSON	JEFF
7	20009800	ADKINSON	PHYLLIS
8	20011000	ADKINSON	BOB
9	20012700	ADKINSON	HAZEL
10	20013800	ADKINSON	DAVID

PLEASE SELECT LINE-NR FOR SALARY INFORMATION: 1

ABELLAN 1450000 PTA

GET SAME

Anmerkung:

Dieses Statement ist nur beim Zugriff auf Adabas- oder VSAM-Datenbanken gültig. Mit Entire System Server ist dieses Statement nicht verfügbar.

Structured-Mode-Syntax

```
GET SAME [(r)]
```

Reporting-Mode-Syntax

```
GET SAME [(r)] [operand1...]
```

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S A	A N P B	nein	ja

Funktion

Das Statement GET SAME dient dazu, einen Datensatz, der gerade verarbeitet wird, erneut zu lesen. Das Statement wird in der Regel dazu verwendet, Werte von Datenbank-Arrays (Periodengruppen oder multiplen Feldern) zu erhalten, falls die Nummer(n) und der Bereich der vorhandenen bzw. gewünschten Ausprägung(en) nicht bekannt war, als der Datensatz zum erstenmal gelesen wurde.

Statement-Referenzierung (r)

Durch Angabe eines Statement-Labels oder der Sourcecode-Zeilenummer (r) können Sie das FIND- oder READ-Statement referenzieren, mit dem der Datensatz zum erstenmal gelesen wurde. Falls keine Referenzierung erfolgt, bezieht sich das GET SAME-Statement auf die innerste aktive Verarbeitungsschleife.

operand1

Als *operand1* geben Sie das Feld bzw. die Felder an, deren Werte Sie mit dem GET SAME-Statement erhalten wollen.

Anmerkung:

Operand1 kann nicht angegeben werden, wenn das Feld in einem DEFINE DATA-Statement definiert ist.

Einschränkungen

Bei einem UPDATE- oder DELETE-Statement darf keine Referenzierung auf ein GET SAME-Statement erfolgen; vielmehr sollten diese Statements das FIND-, READ- oder GET-Statement referenzieren, mit dem der betreffende Datensatz ursprünglich gelesen wurde.

Bei VSAM-Datenbanken gilt GET SAME nur für ESDS und RRDS. Für ESDS muß die RBA in einer numerischen Benutzervariablen enthalten sein oder als Ganzzahl-Konstante angegeben werden. Dasselbe gilt für RRDS, außer daß hier statt der RBA die RRN angegeben werden muß.

Beispiel

```

/* EXAMPLE 'GSAEX1S': GET SAME
/*****
DEFINE DATA LOCAL
1 I          (P3)
1 #NAME      (A30)
1 POST-ADDRESS VIEW OF EMPLOYEES
  2 FIRST-NAME
  2 NAME
  2 ADDRESS-LINE (I:I)
  2 C*ADDRESS-LINE
  2 POST-CODE
  2 CITY
END-DEFINE
/*****
FORMAT PS=20
MOVE 1 TO I
READ POST-ADDRESS BY NAME
  COMPRESS NAME FIRST-NAME INTO #NAME WITH DELIMITER ', '
  WRITE // 12T #NAME
  WRITE / 12T ADDRESS-LINE (I.1)
  IF C*ADDRESS-LINE > 1
    FOR I = 2 TO C*ADDRESS-LINE
      GET SAME                               /* READ NEXT OCCURRENCE
      WRITE 12T ADDRESS-LINE (I.1)
    END-FOR
  END-IF
  WRITE / POST-CODE CITY
  SKIP 3
END-READ
END

```

PAGE	1	97-09-01	11:35:33
	ABELLAN, KEPA		
	CASTELAN 23-C		
28014	MADRID		
	ACHIESON, ROBERT		
	144 ALLESTREE LANE		
	DERBY		
	DERBYSHIRE		
DE3 4TR	DERBY		

GET TRANSACTION DATA

Anmerkung:

Dieses Statement gilt nur für Transaktionen auf Adabas-Datenbanken, sowie auf DL/I-Datenbanken in einer batch-orientierten BMP-Region (nur in IMS-Umgebungen).

GET TRANSACTION [DATA] operand1...

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S	A N P I F B D T	ja	ja

Verwandtes Statement

END TRANSACTION.

Funktion

Das Statement GET TRANSACTION DATA dient dazu, die Transaktionsdaten, die mit einem vorherigen END TRANSACTION-Statement gespeichert wurden, zu lesen.

GET TRANSACTION DATA erzeugt keine Verarbeitungsschleife.

Hinweis für DL/I:

Das GET TRANSACTION DATA-Statement liest Checkpoint-Daten, die mit einem END TRANSACTION-Statement gespeichert wurden.

Systemvariable *ETID

Um die von der Datenbank zu lesenden Transaktionsdaten zu identifizieren, kann die Natural-Systemvariable *ETID eingesetzt werden.

Angabe der Felder (*operand1*)

Reihenfolge, Länge und Format der mit GET TRANSACTION DATA zu lesenden Felder muß mit Reihenfolge, Länge und Format der mit dem jeweiligen END TRANSACTION-Statement angegebenen Felder übereinstimmen.

Hinweis für DL/I:

Der erste operand1 muß eine 8 Byte lange Checkpoint-ID sein.

Keine Transaktionsdaten gespeichert

Werden bei der Ausführung des GET TRANSACTION DATA-Statements keine Transaktionsdaten gefunden, werden alle mit dem Statement angegebenen Felder mit Leerzeichen gefüllt, gleichgültig welches Format die Felder haben. Achten Sie darauf, daß keine arithmetischen Operationen auf der Grundlage "leerer" Transaktionsdaten ausgeführt werden, da dies einen Programmabbruch zur Folge hätte.

Beispiel

```

/* EXAMPLE 'GTREX1S': GET TRANSACTION DATA (STRUCTURED MODE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
  2 MIDDLE-I
  2 CITY
1 #PERS-NR (A8) INIT <' '>
END-DEFINE
/*****
GET TRANSACTION DATA #PERS-NR
IF #PERS-NR NE ' '
  WRITE 'LAST TRANSACTION PROCESSED FROM PREVIOUS SESSION' #PERS-NR
END-IF
/*****
REPEAT
/*****
  INPUT 10X 'ENTER PERSONNEL NUMBER TO BE UPDATED:' #PERS-NR
  IF #PERS-NR = ' '
    STOP
  END-IF
/*****
FIND EMPLOY-VIEW WITH PERSONNEL-ID = #PERS-NR
  IF NO RECORDS FOUND
    REINPUT 'NO RECORD FOUND'
  END-NOREC
  INPUT (AD=M) PERSONNEL-ID (AD=O)
    / NAME
    / FIRST-NAME
    / CITY

  UPDATE
  END TRANSACTION #PERS-NR
END-FIND
/*****
END-REPEAT
END

```

HISTOGRAM

```

HISTOGRAM [ ALL ] [VALUE] [IN] [FILE] view-name
              (operand1)
              [PASSWORD=operand2]
              [IN] { ASCENDING
                    DESCENDING
                    VARIABLE operand3 } [SEQUENCE]
              [VALUE] [FOR] [FIELD] operand4
              [STARTING/ENDING-clause]
              [WHERE logical-condition]
              statement ...
END-HISTOGRAM (nur Structured Mode)
[LOOP]          (nur Reporting Mode)

```

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	N P I	ja	nein
Operand2	C S	A	ja	nein
Operand3	S	A	ja	nein
Operand4	S	A N P I F B D T L	nein	nein

Verwandte Statements

FIND, READ.

Funktion

Das Statement HISTOGRAM dient dazu, Werte eines Datenbankfeldes, das als Deskriptor, Subdeskriptor oder Superdeskriptor definiert ist, zu lesen. Die Werte werden direkt von der Adabas Invertierten Liste (bzw. dem VSAM-Index) gelesen.

Das HISTOGRAM-Statement löst zwar eine Verarbeitungsschleife aus, es kann aber auf keine anderen Datenbankfelder außer des mit dem Statement angegebenen Feldes zugegriffen werden.

Hinweis für SQL-Datenbanken:

Mit HISTOGRAM erhalten Sie die Anzahl der Reihen, die in einer bestimmten Spalte den gleichen Wert haben.

Einschränkungen

Dieses Statement kann nicht für DL/I-Datenbanken oder mit Entire System Server verwendet werden.

Bei einer VSAM-Datenbank gilt das HISTOGRAM-Statement nur für KSDS und ESDS.

Begrenzen der Schleifendurchläufe (*operand1/ALL*)

Sie können die Anzahl der Deskriptorwerte, die mit dem HISTOGRAM-Statement gelesen werden sollen, auf eine bestimmte Zahl (*operand1*) begrenzen, die Sie entweder als numerische Konstante (0 bis 99999999) oder über eine Benutzervariable (die einen Ganzzahlwert enthält) angeben. Andernfalls werden alle Deskriptorwerte gelesen, was Sie zusätzlich durch das Schlüsselwort ALL betonen können.

Das mit *operand1* angegebene Limit hat für dieses Statement Vorrang vor einem mit einem LIMIT-Statement gesetzten Limit.

Ist mit dem LT-Parameter ein kleineres Limit gesetzt, so gilt das LT-Limit.

Anmerkungen:

Wenn Sie eine vierstellige Anzahl von Deskriptorwerten lesen möchten, geben Sie diese mit einer vorangestellten Null an: (0nnnn); denn Natural interpretiert jede vierstellige Zahl in Klammern als Zeilennummer-Referenzierung auf ein Statement.

Operand1 wird zu Beginn des ersten HISTOGRAM-Schleifendurchlaufs ausgewertet. Wird der Wert von operand1 innerhalb der HISTOGRAM-Schleife geändert, hat dies keine Auswirkungen auf die Anzahl der gelesenen Werte.

view-name

Als *view-name* geben Sie den Namen eines Views an, der entweder in einem DEFINE DATA-Block oder in einer programmexternen Global oder Local Data Area definiert ist.

Der View darf außer dem im HISTOGRAM-Statement verwendeten Feld (*operand4*) keine anderen Felder enthalten.

Ist das im View definierte Feld ein in einer Periodengruppe enthaltenes Feld oder multiples Feld, das mit einem Indexbereich definiert ist, dann wird jeweils nur die erste Ausprägung dieses Bereiches vom HISTOGRAM-Statement gefüllt; alle anderen Ausprägungen bleiben von der Ausführung des HISTOGRAM-Statements unberührt.

Im *Reporting Mode* darf *view-name* auch der Name eines DDMs sein.

PASSWORD-Klausel

Die PASSWORD-Klausel dient dazu, ein Paßwort (*operand2*) anzugeben, um auf Daten einer paßwort-geschützten Adabas-Datei zugreifen zu können. Weitere Informationen hierzu siehe FIND-Statement und PASSW-Statement.

SEQUENCE-Klausel

Die Klausel gilt nur für Adabas-, VSAM- und SQL-Datenbanken.

Mit dieser Klausel können Sie bestimmen, ob die Werte in aufsteigender Reihenfolge oder in absteigender Reihenfolge gelesen werden sollen.

- Standardmäßig werden die Werte in aufsteigender Reihenfolge gelesen (was Sie mit dem Schlüsselwort ASCENDING auch ausdrücklich angeben können, aber nicht müssen).
- Wenn die Werte in absteigender Reihenfolge gelesen werden sollen, geben Sie das Schlüsselwort DESCENDING an.
- Wenn erst zur Laufzeit bestimmt werden soll, ob die Werte in aufsteigender oder absteigender Reihenfolge gelesen werden sollen, geben Sie das Schlüsselwort VARIABLE gefolgt von einer Variablen (*operand3*) an. Der Wert von *operand3* zu Beginn der HISTOGRAM-Verarbeitungsschleife bestimmt dann die Reihenfolge. *Operand3* muß Format/Länge A1 haben und kann den Wert "A" (für "aufsteigend") oder "D" (für "absteigend") enthalten.

Anmerkung für Adabas:

Absteigende Lesereihenfolge setzt folgende Adabas-Versionen voraus: Version 3.1 auf UNIX und Windows, Version 3.2 auf OpenVMS bzw. Version 6.2 auf Großrechnern.

Anmerkung für SQL-Datenbanken:

Auf Großrechnern kann die VARIABLE-Option bei SQL-Datenbanken nicht verwendet werden.

Beispiele für SEQUENCE-Klausel:

Siehe Programme HSTDSCND und HSTVSEQ in Library SYSEXRM.

Deskriptor (*operand4*)

Als *operand4* kann ein Deskriptor, ein Subdeskriptor, ein Superdeskriptor oder ein Hyperdeskriptor angegeben werden.

Ein Deskriptorfeld, das Teil einer Periodengruppe ist, kann entweder mit oder ohne Index angegeben werden. Wird kein Index angegeben, so wird ein Datensatz ausgewählt, wenn der Suchwert in irgendeiner Ausprägung gefunden wird. Wird ein Index angegeben, so wird ein Datensatz nur ausgewählt, wenn der Suchwert in der im Index angegebenen Ausprägung gefunden wird. Es muß ein konstanter Index angegeben werden; es darf kein Indexbereich angegeben werden.

Ist das angegebene Deskriptorfeld ein multiples Feld, darf kein Index angegeben werden. Ein Datensatz wird ausgewählt unabhängig davon, in welcher Ausprägung des Feldes der Suchwert gefunden wird.

STARTING/ENDING-clause

$\left[\left[\text{STARTING} \right] \left[\text{WITH FROM} \right] \left[\text{VALUES} \right] \text{operand5} \right] \left[\left[\text{THRU ENDING AT} \right] \text{operand6} \right]$
--

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand5	C S	A N P I F B D T L	ja	nein
Operand6	C S	A N P I F B D T L	ja	nein

Mit den Schlüsselwörtern **STARTING** und **ENDING AT** (bzw. **THRU**) können Sie einen Startwert (*operand5*) und einen Endwert (*operand6*) angeben, und zwar in Form einer Konstanten oder einer Benutzervariablen. Damit legen Sie fest, ab welchem Wert und bis zu welchem Wert gelesen werden soll.

Wenn der angegebene Startwert nicht vorhanden ist, so wird der nächsthöhere vorhandene Wert als Startwert genommen. Ist kein höherer Wert vorhanden, wird die HISTOGRAM-Schleife nicht ausgeführt.

Wenn Sie einen Endwert angeben, wird bis einschließlich des Endwertes gelesen.

Für Deskriptoren des Formats A oder B können hexadezimale Konstanten als Start- und Endwert angegeben werden.

WHERE-Klausel

Mit der WHERE-Klausel können Sie ein zusätzliches Selektionskriterium in Form einer logischen Bedingung (*logical-condition*) angeben. Dies wird ausgewertet, *nachdem* ein Wert gelesen wurde, aber *bevor* eine weitere Verarbeitung auf der Grundlage dieses Wertes (einschließlich AT BREAK-Verarbeitung) erfolgt.

Der in der WHERE-Klausel angegebene Deskriptor muß mit dem im HISTOGRAM-Statement referenzierten Deskriptor identisch sein. Näheres zu logischen Bedingungen finden Sie im Abschnitt **Logische Bedingungen** im *Natural Referenzhandbuch*.

Systemvariablen

Folgende Natural-Systemvariablen können mit einem HISTOGRAM-Statement eingesetzt werden:

- ***NUMBER** — Enthält die Anzahl der Datensätze, die den zuletzt gelesenen Wert enthalten. (Bezüglich SQL-Datenbanken, siehe Beschreibung der Systemvariablen *NUMBER im *Natural Referenzhandbuch*.)
- ***ISN** — Enthält die Nummer der Ausprägung einer Periodengruppe des aktuellen Datensatzes, die den zuletzt gelesenen Wert enthält. Ist der Deskriptor nicht in einer Periodengruppe enthalten, enthält *ISN den Wert "0".
Bei SQL- und VSAM-Datenbanken kann *ISN nicht verwendet werden.
- ***COUNTER** — Enthält die Gesamtzahl der bisher gelesenen Werte (nach Auswertung der WHERE-Klausel).

*ISN und *NUMBER stehen erst nach Auswertung der WHERE-Klausel zur Verfügung. Sie dürfen nicht innerhalb der logischen Bedingung einer WHERE-Klausel eingesetzt werden.

Beispiel

```

/* EXAMPLE 'HSTEX1S': HISTOGRAM (STRUCTURED MODE)
/*****
DEFINE DATA LOCAL
  1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
END-DEFINE
/*****
LIMIT 8
HISTOGRAM EMPLOY-VIEW CITY STARTING FROM 'M'
  DISPLAY NOTITLE CITY
    'NUMBER OF/PERSONS' *NUMBER *COUNTER
END-HISTOGRAM
/*****
END

```

CITY	NUMBER OF PERSONS	CNT
MADISON	3	1
MADRID	41	2
MAILLY LE CAMP	1	3
MAMERS	1	4
MANSFIELD	4	5
MARSEILLE	2	6
MATLOCK	1	7
MELBOURNE	2	8

Äquivalentes Reporting-Mode-Beispiel: siehe Programm HSTEX1R in Library SYSEXRM.

IF

Structured-Mode-Syntax

```
IF logical-condition  
  [THEN] statement...  
  [ELSE statement ...]  
END-IF
```

Reporting-Mode-Syntax

```
IF logical-condition  
  [THEN] { statement  
           DO statement... DOEND }  
  [ELSE { statement  
          DO statement... DOEND }]
```

Verwandte Statements

DECIDE FOR, DECIDE ON.

Funktion

Mit dem Statement IF wird die Verarbeitung eines Statements oder einer Gruppe von Statements in Abhängigkeit von einer logischen Bedingung (*logical-condition*) gesteuert.

Anmerkung:

Falls **keine** Handlung ausgeführt werden soll, wenn die Bedingung erfüllt ist, geben Sie das Statement *IGNORE* in der *THEN*-Klausel an.

logical-condition

Die logische Bedingung, die Sie definieren, bestimmt, wann das Statement bzw. die Statements, die Sie im IF-Statement angeben, ausgeführt werden sollen und wann nicht.

Beispiele:

```
IF #A = #B
```

```
IF LEAVE-TAKEN GT 30
```

```
IF #SALARY(1) * 1.15 GT 5000
```

```
IF SALARY (4) = 5000 THRU 6000
```

```
IF DEPT = 'A10' OR = 'A20' OR = 'A30'
```

Näheres hierzu siehe Abschnitt **Logische Bedingungen** im *Natural Referenzhandbuch*.

THEN

In der THEN-Klausel geben Sie die *statements* an, die ausgeführt werden sollen, wenn die logische Bedingung erfüllt ist.

ELSE

In der ELSE-Klausel geben Sie die *statements* an, die ausgeführt werden sollen, wenn die logische Bedingung *nicht* erfüllt ist.

Beispiel

```

* EXAMPLE 'IFEX1S': IF (STRUCTURED MODE)
*****
DEFINE DATA LOCAL
  1 EMPLOY-VIEW VIEW OF EMPLOYEES
    2 PERSONNEL-ID
    2 NAME
    2 FIRST-NAME
    2 SALARY (1)
    2 BIRTH
  1 VEHIC-VIEW VIEW OF VEHICLES
    2 PERSONNEL-ID
    2 MAKE
  1 #BIRTH (D)
END-DEFINE
*
MOVE EDITED '19450101' TO #BIRTH (EM=YYYYMMDD)
SUSPEND IDENTICAL SUPPRESS
LIMIT 20
*
FND. FIND EMPLOY-VIEW WITH CITY = 'FRANKFURT'
      SORTED BY NAME BIRTH
  IF SALARY (1) LT 40000
    WRITE NOTITLE '*****' NAME 30X 'SALARY LT 40000'
  ELSE
    IF BIRTH GT #BIRTH
      FIND VEHIC-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (FND.)
        DISPLAY (IS=ON) NAME BIRTH (EM=YYYY-MM-DD)
          SALARY (1) MAKE (AL=8)
    END-FIND
  END-IF
END-FIND
END

```

NAME	DATE OF BIRTH	ANNUAL SALARY	MAKE	
BAECKER	1956-01-05	74400	BMW	
***** BECKER				SALARY LT 40000
BLOEMER	1979-11-07	45200	FIAT	
FALTER	1954-05-23	70800	FORD	
***** FALTER				SALARY LT 40000
***** GROTHE				SALARY LT 40000
***** HEILBROCK				SALARY LT 40000
***** HESCHMANN				SALARY LT 40000
HUCH	1952-09-12	67200	MERCEDES	
***** KICKSTEIN				SALARY LT 40000
***** KLEENE				SALARY LT 40000
***** KRAMER				SALARY LT 40000

Äquivalentes Reporting-Mode-Beispiel: siehe Programm IFEX1R in Library SYSEXRM.

IF SELECTION

Structured-Mode-Syntax

```

IF SELECTION [NOT UNIQUE [IN [FIELDS]]] operand1...
    [THEN] statement...
    [ELSE statement ...]
END-IF

```

Reporting-Mode-Syntax

```

IF SELECTION [NOT UNIQUE [IN [FIELDS]]] operand1...
    [THEN] { statement
             DO statement... DOEND }
    [ELSE { statement
            DO statement... DOEND } ]

```

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S A	A L C	ja	nein

Funktion

Das Statement IF SELECTION dient dazu, zu verifizieren, daß in einer Reihe von alphanumerischen Feldern genau ein Feld einen Wert enthält.

Die in der THEN-Klausel angegebenen Statements werden ausgeführt, wenn eine der beiden folgenden Bedingungen erfüllt ist:

- Keines der angegebenen Felder (*operand1*) enthält einen Wert.
- Mehr als eines der angegebenen Felder (*operand1*) enthält einen Wert.

Dieses Statement wird in der Regel dazu eingesetzt, zu verifizieren, daß auf einer über ein INPUT-Statement erzeugten Map vom Terminal-Benutzer nicht gleichzeitig mehr als eine Funktion eingegeben wurde.

Anmerkung:

Falls keine Handlung ausgeführt werden soll, wenn eine der beiden Bedingungen erfüllt ist, geben Sie das Statement IGNORE in der THEN-Klausel an.

Selektionsfeld (*operand1*)

Als *operand1* geben Sie die Felder an, die verifiziert werden sollen.

Wenn Sie eine Kontrollvariable (Format C) angeben, so wird angenommen, daß sie einen Wert enthält, wenn ihr Status sich auf “MODIFIED” geändert hat.

Beispiel

```

/* EXAMPLE 'IFSEL': IF SELECTION
/*****
DEFINE DATA LOCAL
  1 #A (A1)
  1 #B (A1)
END-DEFINE
/*****
INPUT 'SELECT FUNCTION:' //
  10X 'READ EMPLOYEES FILE:' #A
  10X 'READ VEHICLES FILE: ' #B
/*****
IF SELECTION NOT UNIQUE #A #B
  REINPUT 'PLEASE ENTER ONE FUNCTION ONLY:'
END-IF
/*****
IF #A NE ' '
  FETCH 'READEMPL'
END-IF
IF #B NE ' '
  FETCH 'READVEHC'
END-IF
/*****
END

```

SELECT FUNCTION:

READ EMPLOYEES FILE: x

READ VEHICLES FILE: x

PLEASE ENTER ONE FUNCTION ONLY:

SELECT FUNCTION:

READ EMPLOYEES FILE: x

READ VEHICLES FILE: x

IGNORE

IGNORE

Funktion

Das Statement IGNORE ist ein “leeres” Statement, das selbst keine Funktion ausführt.

Während der Entwicklungsphase einer Anwendung können Sie IGNORE vorübergehend innerhalb von Statement-Blöcken einsetzen, in denen ein oder mehrere Statements angegeben werden müssen, welche Sie aber erst später codieren möchten (z.B. in AT BREAK oder AT START/END OF DATA). Sie können dann die Programmierung in einem anderen Teil Ihrer Anwendung fortsetzen, ohne daß der noch unvollständige Statement-Block zu einem Fehler führt.

Das IGNORE-Statement kann auch in Bedingungs-Statements wie IF oder DECIDE FOR verwendet werden, wenn nichts ausgeführt werden soll, falls eine bestimmte Bedingung erfüllt ist.

Beispiel

```
...
...
AT TOP OF PAGE
  IGNORE          /* top-of-page processing still to be coded
END-TOPPAGE
...
...
```

INCLUDE

INCLUDE <i>copycode-name</i> [<i>operand1...99</i>]
--

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C	A	nein	nein

Funktion

Das Statement INCLUDE dient dazu, den Sourcecode eines externen Objekts vom Typ Copycode bei der Kompilierung in ein anderes Objekt einzufügen.

Das INCLUDE-Statement wird bei der *Kompilierung* ausgewertet. Die Sourcecode-Zeilen des Copycodes werden nicht physisch in den Sourcecode des Programms eingefügt, das das INCLUDE-Statement enthält, und der eingefügte Copycode ist als Teil des Objektmoduls im kompilierten Programm enthalten.

Eine Sourcecode-Zeile, die ein INCLUDE-Statement enthält, darf kein anderes Statement enthalten.

copycode-name

Als *copycode-name* geben Sie den Namen des Copycodes an, dessen Source eingefügt werden soll.

Der *copycode-name* kann ein Kaufmännisches Und (&) enthalten; zur Kompilierungszeit wird dieses Zeichen durch den aktuellen Wert der Natural-Systemvariablen *LANGUAGE ersetzt. Diese Funktion ermöglicht die Verwendung mehrsprachiger Copycode-Namen.

Das Objekt, dessen Namen Sie angeben, muß vom Objekttyp Copycode sein. Der Copycode muß entweder in derselben Library gespeichert sein wie das Programm, das das INCLUDE-Statement enthält, oder in der betreffenden Steplib (Standard-Steplib ist SYSTEM).

Wenn der Sourcecode des Copycodes verändert wird, müssen alle Programme, in denen der Copycode eingefügt ist, neu kompiliert werden, damit die Änderungen in den Objektmodulen zum Tragen kommen. Der Sourcecode des Copycodes muß aus syntaktisch vollständigen Statements bestehen.

operand1

In den einzufügenden Copycode können Sie dynamisch Werte einsetzen. Diese Werte werden mit *operand1* angegeben.

Im Copycode werden die Werte mit der Notation “&n&” referenziert, d.h. Sie markieren die Stelle, an der ein Wert eingesetzt werden soll, mit “&n&”. “n” ist die laufende Nummer des mit dem INCLUDE-Statement übergebenen Wertes. Zum Beispiel würde sich “&3&” auf den dritten übergebenen Wert beziehen.

Für jede “&n&”-Notation im Copycode müssen Sie im INCLUDE-Statement einen Wert angeben. Wenn der Copycode beispielsweise “&5&” enthält, muß *operand1* mindestens fünfmal angegeben werden.

Mit dem INCLUDE-Statement angegebene Werte, die im Copycode nicht referenziert werden, werden ignoriert.

Anmerkung:

Operand1 selbst darf keine “&n&”-Notation sein.

Beispiel 1

```
/* EXAMPLE 'INCEX1:' INCLUDE
/*****
/* ...
/* ...
/* ...
/*****
INCLUDE MEM1
/*****
/* ...
/* ...
/* ...
END
```

Beispiel 2

Einzufügender Copycode:

```

/* EXAMPLE 'COPEX1': COPYCODE USING PARAMETERS
READ (&4&) &1& BY &2& = &3&
  DISPLAY &2&
  IF &2& = &5& DO
    WRITE 5X 'LAST RECORD FOUND'
    STOP
  DOEND
LOOP

```

Programm mit INCLUDE-Statement:

```

/* EXAMPLE 'COPEX2': PROGRAM USING COPYCODE WITH PARAMETERS
*
INCLUDE COPEX1 'EMPLOYEES' 'NAME' '''ALDEN''' '20' '''ALLEN'''
END

```

Page	1	91-06-18 14:01:26
<hr/> LAST-NAME		
ALDEN ALEXANDER ALEXANDER ALEXANDER ALEXANDER ALEXANDER ALEXANDER ALEXANDER ALEXANDER ALEXANDER ALLDERIDGE ALLDERIDGE ALLDERIDGE ALLEN LAST RECORD FOUND		

Beispiel 3

Einzufügender Copycode 1:

```
/* EXAMPLE 'Object ASSIGN':
   &1& := &2&
```

Einzufügender Copycode 2:

```
/* EXAMPLE 'Object ASET':
   INCLUDE ASSIGN &1& := &2&
```

INCLUDE-Statement enthaltendes Programm

```
/* EXAMPLE PROGRAM USING COPYCODE WITH PARAMETERS
*
  reset #a (i4)
  move 123 to #a
  write '=' #a
  include ASSIGN '#a' '5'
  write '=' #a
  move 123 to #a
  write '=' #a
  include ASET '''#a''' '''5'''
  write '=' #a
  end
```

Page 1 02-01-30 14:01:26

```
#A: 123
#A: 5
#A: 123
#A: 5
```

INPUT

Verwandtes Statement

REINPUT.

Funktion

Das INPUT-Statement dient bei der interaktiven Verarbeitung dazu, formatierte Schirme oder Maps auszugeben oder zu generieren, die zur Eingabe von Daten verwendet werden.

Das Statement kann auch in Verbindung mit dem Natural-Stack (siehe STACK-Statement) verwendet werden sowie auf Großrechnern zur Eingabe von Benutzerdaten bei Programmen, die im Batch-Betrieb ausgeführt werden.

Eingabe-Modi

Das INPUT-Statement kann unter drei verschiedenen Eingabe-Modi verwendet werden: Screen-Modus, Forms-Modus oder Keyword/Delimiter-Modus. Im Falle von Videoterminals/ Videobildschirmen wird in der Regel der Screen-Modus verwendet. Forms-Modus kann bei TTY-Terminals verwendet werden. Keyword/Delimiter-Modus kann bei TTY-Terminals oder (auf Großrechnern) im Batch-Betrieb benutzt werden. Standardmäßig gilt Screen-Modus.

Sie können den Eingabemodus mit dem Session-Parameter IM oder den Terminalkommandos %F und %D ändern.

Screen-Modus

Im Screen-Modus bewirkt die Ausführung eines INPUT-Statements die Anzeige eines Schirms (Screen) entsprechend der angegebenen Felder und ihrer Positionen. Die Meldungszeile des Schirms wird von Natural zur Ausgabe von Fehlermeldungen benutzt. Die Position dieser Zeile (Kopf- oder Fußzeile) kann mit dem Terminalkommando %M beeinflusst werden. Der Terminal-Benutzer kann über die verschiedenen Tabulator-Tasten bestimmte Felder ansteuern.

Da Natural die sogenannte Bildschirmfenster- oder "Window"-Technik unterstützt, ist es erlaubt, daß die Größe einer logischen vom Programm ausgegebenen Map (theoretisch maximal 250 Stellen breit und 250 Zeilen lang, aber begrenzt durch den internen Bildschirm-Puffer) über die Größe des physischen Bildschirms hinausgeht.

Um ein Bildschirmfenster, d.h. den auf dem physischen Schirm sichtbaren Ausschnitt einer logischen Programmseite, zu beeinflussen und auf der logischen Seite zu verschieben, kann das Terminalkommando %W verwendet werden (Näheres zur Fenster-Verarbeitung siehe Terminalkommando %W).

Für Eingabefelder (AD=A oder AD=M), die auf dem physischen Bildschirm nicht vollständig angezeigt werden, gilt folgendes:

- Ein Eingabefeld, dessen Anfang außerhalb des Fensters liegt, wird immer zu einem geschützten Feld gemacht.
- Ein Eingabefeld, das im Fenster beginnt aber außerhalb des Fensters endet, wird nur dann geschützt, wenn der Wert, den es enthält, nicht vollständig innerhalb des Fensters sichtbar ist. Bitte beachten Sie, daß es hierbei darauf ankommt, ob die *Wertlänge*, nicht die *Feldlänge*, über das Fenster hinausgeht. Füllzeichen (wie mit dem Profilparameter FC oder dem Session-Parameter AD angegeben) zählen nicht als Teil des Wertes.
- Falls Sie in ein derart geschütztes Eingabefeld Eingaben machen möchten, müssen Sie zunächst die Fenstergröße so ändern, daß sich der Anfang des Feldes bzw. das Ende des Feldwertes innerhalb des Fensters befindet (siehe Terminalkommando %W).

Andere Eingabe-Modi

Das INPUT-Statement kann sowohl für Operationen auf zeilenorientierten Geräten wie zur Verarbeitung von Batch-Eingaben aus sequentiellen Dateien verwendet werden.

Dieselben Maps, die im interaktiven Screen-Modus verwendet werden, können auch in einem der anderen Eingabe-Modi verarbeitet werden.

Im Forms- oder Keyword/Delimiter-Modus werden die Eingaben entweder ohne Maps verarbeitet oder durch Map-Simulation im Line-Modus.

Näheres hierzu finden Sie unter **INPUT-Statement unter Nicht-Screen-Modi** (Seite 395), **INPUT-Statement im Batch-Betrieb auf Großrechnern** (Seite 397) und **Input-Daten aus dem Natural-Stack** (Seite 396).

Eingabe von Daten als Reaktion auf ein INPUT-Statement

Bei alphanumerischen Feldern müssen die Daten linksbündig eingegeben werden; jedes eingegebene Zeichen (auch Leerzeichen) hat eine Bedeutung. Die Daten werden ein Zeichen pro Byte dem internen Feld zugeordnet. In ein alphanumerisches Feld eingegebene Daten werden nicht auf Gültigkeit überprüft.

Die Umsetzung von Klein- in Großbuchstaben kann über die Terminalkommandos %L und %U sowie die Feldattribute AD=T und AD=W gesteuert werden.

In numerische Felder können Daten an beliebiger Stelle eingegeben werden, wobei Leerzeichen und Nullen vor und Leerzeichen nach dem eingegebenen Wert erlaubt sind; darüber hinaus dürfen ein Vorzeichen und ein Komma (Dezimalpunkt) eingegeben werden. Natural richtet den Feldwert entsprechend der internen Definition des Feldes aus.

Gilt SG=OFF, so reserviert oder vergibt Natural keine Stelle für das Vorzeichen. Bei Feldern mit Format P müssen Daten in Dezimalform eingegeben werden; falls nötig, setzt Natural dezimale Daten automatisch in gepackte um. Ein Feld, das nur Leerzeichen enthält, wird als Nullwert interpretiert.

Bei in ein numerisches Feld eingegebenen Daten überprüft Natural, ob es sich um keine anderen Zeichen als Zahlenzeichen, Komma (Dezimalpunkt), Vorzeichen (optional) und vor- oder nachgestellte Leerzeichen handelt. Wird kein Komma eingegeben, so wird angenommen, daß es sich rechts neben dem eingegebenen Wert befindet.

Daten für binäre Felder müssen für alle Byte-Positionen eingegeben werden (zwei Zeichen pro Byte); es dürfen nur Hexadezimalzeichen (0 – 9, A – F) eingegeben werden. Ein Leerzeichen (H'20' in ASCII bzw. H'40' in EBCDIC) ist erlaubt und wird in binäre Nullen umgesetzt. Natural überprüft, ob keine anderen außer den gültigen Hexadezimalzeichen eingegeben wurden.

Bei logischen Feldern (Format L) kann entweder ein Leerzeichen (für "Falsch") oder ein anderes Zeichen (für "Wahr") eingegeben werden.

Bei Feldern der Formate F, D und T müssen die Daten entsprechend den für Gleitkomma-, Datums- bzw. Zeitkonstanten gültigen Regeln eingegeben werden.

Eingabefehler

Entsprechen die in ein Eingabefeld eingegebenen Daten nicht dem Format bzw. der Editiermaske des Feldes, gibt Natural eine entsprechende Fehlermeldung aus (ohne die Programmausführung abzubrechen) und plaziert den Cursor in das betreffende Feld; der Benutzer kann dann den Fehler berichtigen und gültige Daten eingeben, woraufhin die Verarbeitung fortgesetzt wird.

Split Screen

In der Regel erzeugt jedes INPUT-Statement eine neue Ausgabeseite (bzw. einen neuen Schirm). Ein an ein AT END OF PAGE-Statement geknüpftes INPUT-Statement erzeugt keinen neuen Schirm. Dadurch besteht die Möglichkeit, einen geteilten Schirm (Split Screen) zu erhalten, dessen obere Hälfte mehrere Zeilen anzeigt, während in der unteren Hälfte eine Eingabe-Map erstellt werden kann. Damit die Eingabe-Map auf denselben physischen Schirm paßt, muß die logische Seitenlänge mit dem Parameter PS in einem SET GLOBALS- oder FORMAT-Statement entsprechend gesetzt werden.

Die erste INPUT-Zeile wird unter die letzte angezeigte Zeile plaziert. Falls die NO ERASE-Option verwendet wird, wird die erste INPUT-Zeile an den Anfang der Seite plaziert.

Syntax 1 — Dynamisch generierter Eingabeschirm

INPUT [WINDOW='window-name'] [NO ERASE] [(statement-parameters)] [WITH-TEXT-option] [MARK-option] [ALARM-option] $\left[\begin{array}{l} nX \\ nT \\ x/y \end{array} \right] \left[\begin{array}{l} 'text' [(attributes)] \\ 'c'(n) [(attributes)] \\ \text{'_'} \\ \text{'='} \\ \text{'/'...} \end{array} \right] \left[\begin{array}{l} *IN \\ *OUT \\ *OUTIN \end{array} \right] \{operand1 [(parameter)]\} \dots$				
---	--	--	--	--

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S A G N	A N P I F B D T L G	ja	ja

Diese Form des INPUT-Statements wird dazu verwendet, entweder einen Eingabe-Schirm zu generieren oder ein Eingabedaten-Layout zu erstellen, das (auf Großrechnern) im Batch-Betrieb von einer sequentiellen Eingabedatei gelesen werden kann.

INPUT WINDOW='window-name'

Mit der Option WINDOW='window-name' bewirken Sie, daß das INPUT-Statement für das angegebene Fenster (Window) ausgeführt werden soll. Das angegebene Fenster muß in einem DEFINE WINDOW-Statement definiert sein. Das angegebene Fenster ist nur für die Dauer des betreffenden INPUT-Statements aktiv und wird nach Ausführung des INPUT-Statements automatisch deaktiviert.

Vgl. Statements DEFINE WINDOW und SET WINDOW).

NO ERASE

NO ERASE bewirkt, daß die vom INPUT-Statement ausgegebene Schirmanzeige eine bereits vorhandene Anzeige überlagern kann, ohne letztere zu löschen.

Schirm bezieht sich in diesem Zusammenhang auf die logische Ausgabe und nicht auf den physischen Bildschirm.

Alle ungeschützten Felder auf dem alten Schirm werden geschützt, so daß keine Eingaben mehr in sie gemacht werden können. Die alten Daten bleiben auf dem Schirm, bis der neue Schirm angezeigt wird. Überlagert ein Feld des neuen Schirms teilweise ein altes, so werden das Zeichen vor dem neuen Feld und das nächste Zeichen im alten Feld durch ein Leerzeichen ersetzt.

statement-parameters

Unmittelbar nach dem Schlüsselwort INPUT oder nach einem der auszugebenden Felder können Sie in Klammern Session-Parameter setzen.

Diese Parameter haben dann für das jeweilige Statement oder Feld Gültigkeit statt der betreffenden mit einem GLOBALS-Kommando, SET GLOBALS- oder FORMAT-Statement gesetzten Parameter. Werden mehrere Parameter angegeben, müssen sie jeweils durch ein oder mehrere Leerzeichen voneinander getrennt werden. Die Angabe eines Parameters darf sich nicht über zwei Sourcecode-Zeilen erstrecken.

Die hier gültigen Parameter-Einstellungen kommen nur für Variablen-Felder in Betracht, haben aber keine Auswirkungen auf Text-Konstanten. Wenn Sie Feldattribute für eine Text-Konstante setzen möchten, dann müssen Sie explizit für dieses Element gesetzt werden.

Beispiel

```

DEFINE DATA LOCAL
1 VARI (A4)      INIT <'1234'>          /*   Displays
END-DEFINE      /*   as
FORMAT AD=M     /*   _____
INPUT           'Text'                VARI          /*   Text 1234
INPUT (PM=I)    'Text'                VARI          /*   Text 4321
INPUT          'Text' (PM=I)          VARI (PM=I)   /*   txeT 4321
INPUT          'Text' (PM=I)          VARI          /*   txeT 1234
END

```

Beispiele für den Einsatz von Parametern auf Statement- und Element-Ebene finden Sie auf den folgenden Seiten. Welche Parameter Sie verwenden dürfen, sowie eine Beschreibung der einzelnen Parameter siehe Kapitel **Session-Parameter** im *Natural Referenzhandbuch*.

WITH TEXT-option

[WITH] TEXT $\left[\begin{array}{l} *operand1 \\ operand2 \end{array} \right]$ [(attributes)] [,operand3] ...7
--

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	N P I B	ja	nein
Operand2	C S	A	ja	nein
Operand3	C S	A N P I F B D T L	ja	nein

Diese Option dient dazu, Text anzugeben, der in der Meldungszeile ausgegeben werden soll. In der Regel handelt es sich dabei um eine Meldung, was auf dem jeweiligen Schirm getan werden soll bzw. wie eine falsche Eingabe korrigiert werden soll.

Meldungstext aus der Natural-Fehlermeldungsdatei (*operand1)

Als *operand1* geben Sie eine Natural-Fehlernummer an. Natural liest dann die entsprechende Fehlermeldung von der Natural-Fehlermeldungsdatei.

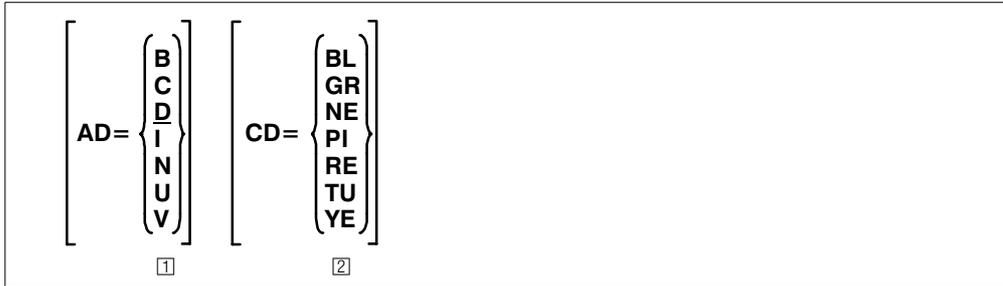
In der Fehlermeldungsdatei sind die Fehlermeldungen nach Libraries sortiert gespeichert. Pro Library können bis zu 9999 Meldungen gespeichert werden.

Näheres zu Fehlermeldungen finden Sie im Abschnitt **Message and Text Files** unter **Generating Messages Modules** in der *Natural SYSERR Utility Documentation*.

Meldungstext (*operand2*) und Attribute (*attributes*)

Als *operand2* geben Sie den Text an, der in der Meldungszeile ausgegeben werden soll.

Als *attributes* können Sie *operand1* oder *operand2* bestimmte Anzeige- und Farbattribut zuordnen. Sie können die folgenden *attributes* angeben:



- ① Anzeigeattribute (siehe Session-Parameter AD im *Natural Referenzhandbuch*).
- ② Farbattribute (siehe Session-Parameter CD im *Natural Referenzhandbuch*).

Dynamische Meldungstext-Komponente (*operand3*)

Operand3 kann in Form einer numerischen Konstanten oder Textkonstanten oder als Name einer Variablen angegeben werden.

Der angegebene Wert dient dazu, einen Teil der Meldung dynamisch zu generieren.

Innerhalb der Fehlermeldung dient die Notation “:n:” zur Referenzierung von *operand3*, wobei *n* die Ausprägung (1 – 7) von *operand3* darstellt.

Beispiel:

```
...
MOVE 'MESSAGE-1' TO #FIELD
...
INPUT WITH TEXT 'THE ERROR IS :1: ',#FIELD ...
...
```

Dies würde die Ausgabe folgender Meldung bewirken:

THE ERROR IS MESSAGE-1

Anmerkung:

Wird *operand3* mehrmals angegeben, müssen diese Operanden mit einem Komma voneinander getrennt werden. Falls das Komma als Dezimalzeichen verwendet wird (wie mit dem Session-Parameter *DC* definiert) und es sich bei *operand3* um numerische Konstanten handelt, setzen Sie Leerzeichen vor und nach dem Komma, damit es nicht als Dezimalkomma mißinterpretiert wird.

Alternativ können mehrere Ausprägungen von *operand3* auch mit dem Input-Delimiterzeichen (wie mit dem Session-Parameter *ID* definiert) voneinander getrennt werden; dies geht jedoch nicht bei *ID=/* (Schrägstrich), da der Schrägstrich in der Syntax des *INPUT*-Statements eine andere Bedeutung hat.

Nicht signifikante Nullen oder Leerzeichen werden aus dem Feldwert entfernt, bevor er in einer Meldung angezeigt wird.

MARK-option

MARK [POSITION <i>operand4</i> [IN]] [FIELD] { <i>operand1</i> * <i>fieldname</i> }
--

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand4	C S	N P I	ja	nein
Operand1	C S A	N P I	ja	nein

Das zu markierende Feld (*operand1*)

Mit der Option MARK FIELD markieren Sie ein bestimmtes Feld, d.h. bei der Ausführung des INPUT-Statements wird der Cursor in dieses Feld plaziert. Jedes mit einem INPUT-Statement angegebene Eingabefeld (AD=A oder AD=M) wird durchnummeriert, beginnend mit 1. Sie können entweder als *operand1* die Nummer des Feldes oder mit der Notation **fieldname* den Namen des Feldes angeben, das mit dem Cursor markiert werden soll.

MARK POSITION

Mit MARK POSITION können Sie den Cursor an eine bestimmte Stelle — die Sie mit *operand4* angeben — innerhalb des Feldes plazieren. *Operand4* darf keine Dezimalstellen enthalten.

Beispiele:

```
MARK 3
MARK #A
MARK *#
MARK POSITION 3 IN #A
```

ALARM-option

```
[ [AND] [SOUND] ALARM ]
```

Diese Option bewirkt, daß der Warnton des Terminals ausgelöst wird, wenn das INPUT-Statement ausgeführt wird. Voraussetzung ist, daß die verwendete Terminal-Hardware dies ermöglicht.

Text vor einem Feld

Ist der Session-Parameter IP nicht auf IP=OFF gesetzt, so wird der jeweilige Feldname vor dem Feldwert (Forms-Modus) oder als Aufforderung, ein Feld auszuwählen, (Keyword/Delimiter-Modus) angezeigt. Wenn Sie vor dem Feld '-' angeben, wird der Feldname nicht angezeigt; wenn Sie einen *'text'* angeben, wird dieser statt des Feldnamens angezeigt.

Feldpositionierung, Text, Attributzuweisung

$$\left[\begin{array}{l} n\mathbf{X} \\ n\mathbf{T} \\ x/y \end{array} \right] \left[\begin{array}{l} \text{'text' [(attributes)]} \\ \text{'c'(n) [(attributes)]} \\ \text{'_'} \\ \text{'='} \\ \text{'/...'} \end{array} \right] \left[\begin{array}{l} *IN \\ *OUT \\ *OUTIN \end{array} \right] \{operand1 [(parameter)]\} \dots$$

Verschiedene Notationen stehen zur Feldpositionierung, Texterstellung und Attributzuweisung zur Verfügung.

$n\mathbf{X}$

Fügt zwischen den Feldern n Leerstellen ein.

Anmerkung:

(nur für Großrechner) Diese Notation fügt zwischen den Spalten n Leerstellen ein. n darf nicht "0" sein.

$n\mathbf{T}$

Setzt einen Tabulator, d.h. die Ausgabe eines Feldes beginnt ab Spalte n .

x/y

Gibt das nachfolgende Element in Zeile x ab Spalte y aus. y darf nicht "0" sein. Rückwärtspositionierung in derselben Zeile ist nicht möglich.

'text'

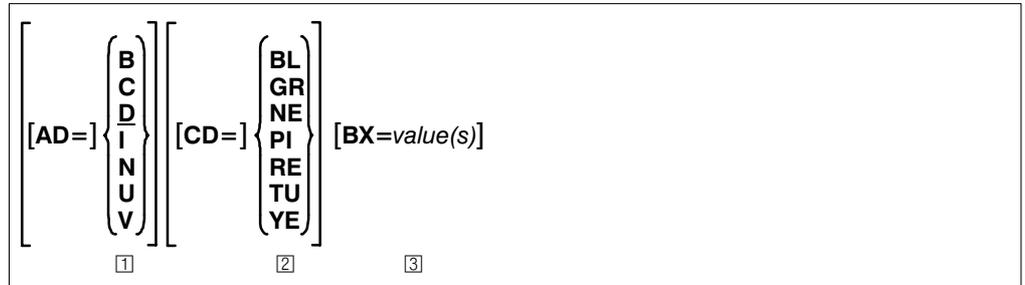
In Apostrophen angegebener *text* wird schreibgeschützt ausgegeben.

'c'(n)

Wie 'text'. Aber das Zeichen c wird n -mal ausgegeben. n darf 1 – 132 sein.

attributes

Dient dazu, den ausgegebenen Feldern/Textelementen Anzeige- und Farbattribute zuzuordnen. Sie können die folgenden *attributes* angeben:



- ① Anzeigeattribute (siehe Session-Parameter AD im *Natural Referenzhandbuch*).
- ② Farbattribute (siehe Session-Parameter CD im *Natural Referenzhandbuch*).
- ③ “Outlining”-Attribute (siehe Session-Parameter BX im *Natural Referenzhandbuch*).

Minuszeichen ‘-’

Diese Notation unmittelbar vor einem Feld bewirkt, daß die Anzeige des Feldnamens vor dem Feld unterdrückt wird.

Anmerkung:

Eine Textkette vor einem Feld ersetzt den Feldnamen als Eingabeaufforderungstext.

Gleichheitszeichen ‘=’

Diese Notation unmittelbar vor einem Feld bewirkt, daß unmittelbar vor dem Feldwert die Feldüberschrift ausgegeben wird.

Schrägstrich /

Ein Schrägstrich zwischen zwei Feldern oder Textelementen bewirkt einen Zeilenvorschub, d.h. das nachfolgende Element wird in der nächsten Zeile ausgegeben.

Felder können als reine Eingabefelder (AD=A), reine Ausgabefelder (AD=O) oder als modifizierbare Ausgabefelder (AD=M) definiert werden. Standardmäßig gilt AD=A. Felder, die mit AD=A oder AD=M definiert sind, werden als ungeschützte Felder ausgegeben, d.h. der Benutzer hat die Möglichkeit, Daten in diese Felder einzugeben.

Bei TTY-Geräten beansprucht die Ausgabe modifizierbarer Felder die doppelte Feldgröße (einmal für Ausgabe und einmal für Eingabe), damit ein neuer Wert eingegeben werden kann. Bei TTY-Bildschirmen beginnt ein Eingabefeld (AD=A oder AD=M), dessen Wert bei der Eingabe nicht angezeigt wird (Feldattribut N), immer in einer neuen Zeile.

Beispiel:

INPUT #A (AD=A) #B (AD=O) #C (AD=M)

#A ist ein Eingabefeld (ungeschützt), in das ein Wert eingegeben werden kann.

#B ist ein Ausgabefeld (schreibgeschützt), dessen angezeigter Wert nicht überschrieben werden kann.

#C ist ein Feld, dessen ausgegebener Wert verändert werden kann, indem er durch einen neuen Wert überschrieben wird.

***IN *OUT *OUTIN**

Entspricht den Feldattributen AD=A bzw. AD=O bzw. AD=M.

Feldangabe (*operand1*)

Als *operand1* geben Sie das zu verwendende Feld an. Sie können Datenbankfelder oder Benutzervariablen angeben.

Natural überträgt den Inhalt eines Feldes direkt vom Datenbereich an das INPUT-Statement; eine MOVE-Operation ist hierzu nicht erforderlich.

Ändert sich der Wert eines Datenbankfeldes aufgrund einer INPUT-Verarbeitung, so betrifft dies nur den Feldwert im Datenbereich. Um den Wert eines Feldes auf der Datenbank zu ändern, sind entsprechende UPDATE-/STORE-Statements erforderlich.

Wird in einem INPUT-Statement der Name einer Gruppe von Datenbankfeldern referenziert, so werden alle in der Gruppe enthaltenen Felder einzeln als Eingabefelder verwendet.

Wird ein Bereich von Ausprägungen eines Arrays referenziert, so wird jede Ausprägung einzeln als Eingabefeld verarbeitet; allerdings wird nur der ersten Ausprägung ein Text oder der Feldname vorangestellt.

parameters

Unmittelbar nach *operand1* können Sie in Klammern einen oder mehrere Parameter angeben.

Diese Parameter haben dann für das jeweilige Feld Gültigkeit statt der betreffenden mit einem GLOBALS-Kommando, SET GLOBALS- oder FORMAT-Statement gesetzten Parameter. Werden mehrere Parameter angegeben, müssen sie jeweils durch ein oder mehrere Leerzeichen voneinander getrennt werden. Die Angabe eines Parameters darf sich nicht über zwei Sourcecode-Zeilen erstrecken.

Anmerkung:

Ist für ein Datenbankfeld eine Editiermaske definiert, so wird der Editiermasken-Parameter EM dynamisch im entsprechenden DDM referenziert. Editiermasken können für Eingabe- wie für Ausgabefelder angegeben werden. Wird für ein Eingabefeld eine Editiermaske definiert, müssen die Daten in Einklang mit der Editiermasken-Definition eingegeben werden.

Beispiel 1 (Verwendung von Syntax 1)

```

/* EXAMPLE 'IPTEX1': INPUT
/*****
DEFINE DATA LOCAL
1 #PNUM (A8)
1 #FNC (A1)
END-DEFINE
/*****
INPUT 10X 'SELECTION MENU FOR EMPLOYEES SYSTEM' /
      10X '-' (54) //
      10X 'ADD ' '(A)' /
      10X 'PURGE' '(P)' /
      10X 'UPDATE' '(U)' /
      10X 'TERMINATE' '(A)' ///
      10X 'PLEASE ENTER FUNCTION: ' #FNC
/*****
DECIDE ON EVERY VALUE OF #FNC
  VALUE 'A'
    WRITE 'Add function selected'
    /* invoke the object containing the Add function here
  VALUE 'P'
    WRITE 'Purge function selected'
    /* invoke the object containing the Purge function here
  VALUE 'U'
    WRITE 'Update function selected'
    /* invoke the object containing the Update function here
  VALUE '.'
    STOP
  NONE
    REINPUT 'PLEASE ENTER A VALID FUNCTION' MARK *#FNC
END-DECIDE
/*****
END

```

```
SELECTION MENU FOR EMPLOYEES SYSTEM
```

```
ADD (A)
PURGE (P)
UPDATE (U)
TERMINATE (.)
```

```
PLEASE ENTER FUNCTION:
```

Beispiel 2 (Verwendung von Syntax 1)

```

/* EXAMPLE 'INPEX1': INPUT WINDOW
*
DEFINE WINDOW WIND1
  SIZE 10 * 40
  BASE 5 / 10
  FRAMED ON POSITION TEXT
*
INPUT WINDOW='WIND1' 'PLEASE ENTER HERE:'
  / #STRING(A15)
*
END

```

```

> r                                     > + Program   INPEX1   Lib SYSEXRM
All  .....1.....2.....3.....4.....5.....6.....7..
0010 /* EXAMPLE 'INPEX1': INPUT WINDOW
0020 *
0030 D +-----Top+
0040 ! PLEASE ENTER HERE:           !
0050 ! #STRING                       !
0060 !                               !
0070 * !                             !
0080 I !                             !
0090 !                               !
0100 * !                             !
0110 E !                             !
0120 +-----Bottom+
0130
0140
0150
0160
0170
0180
0190
0200
.....1.....2.....3.....4.....5..... S 11   L 1

```

Beispiel 3 (Verwendung von Syntax 1)

```
/* EXAMPLE 'INPEX2': INPUT WINDOW
*
ASSIGN #START (A30) = 'EXAM_'
*
INPUT (AD=M) MARK POSITION 5 IN *#START
  / 'PLEASE COMPLETE START VALUE FOR SEARCH'
  / 5X #START
END
```

```
PLEASE COMPLETE START VALUE FOR SEARCH
#START EXAM [ ]
```

Syntax 2 — Verwendung einer vordefinierten Map

```
INPUT [WINDOW='window-name'] [WITH TEXT-option]
```

```
    [MARK-option]
```

```
    [ALARM-option]
```

```
    [USING] MAP map-name [NO ERASE]
```

```
    [ operand1... ]
    [ NO PARAMETER ]
```

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Map-name	C S	A	ja	nein
Operand1	C S A	A N P I F B D T L C	ja	ja

Diese Form des INPUT-Statements wird benutzt, wenn bei der Eingabeverarbeitung eine mit dem Natural-Map-Editor erstellte Eingabe-Map verwendet werden soll.

Hierbei gibt es zwei Möglichkeiten:

- das Programm enthält keine Parameterliste
- das Programm enthält eine Parameterliste (*operand1*).

INPUT USING MAP ohne Parameterliste

In diesem Fall muß der *map-name* als alphanumerische Konstante (maximal 8 Zeichen lang) angegeben werden.

Die verwendete Map muß bereits erstellt sein, bevor das Programm, das sie referenziert, kompiliert werden kann.

Die Feldnamen werden bei der Kompilierung dynamisch von der Map-Source übernommen. Die Feldnamen müssen in Map und Programm identisch sein.

Zu diesem Zeitpunkt muß auf alle im INPUT-Statement referenzierten Felder zugegriffen werden können.

Im *Structured Mode* müssen die Felder vorher definiert werden, und Datenbankfelder müssen durch Referenzierung der betreffenden Verarbeitungsschleife bzw. des betreffenden Views korrekt referenziert werden. Im *Reporting Mode* müssen Benutzervariablen in der Map neu definiert werden.

Wird das Layout der Map verändert, müssen die die Map verwendenden Programme nicht neu katalogisiert werden. Wenn aber Array-Strukturen oder -Namen, Format/Länge von Feldern geändert oder Felder zur Map hinzugefügt bzw. aus ihr gelöscht werden, müssen die die Map verwendenden Programme neu katalogisiert werden.

Die Map-Source muß bei der Programm-Kompilierung zur Verfügung stehen; sonst kann das INPUT USING MAP-Statement nicht kompiliert werden. Wollen Sie das Programm kompilieren, obwohl noch keine Map zur Verfügung steht, geben Sie NO PARAMETER an: das INPUT USING MAP-Statement kann dann kompiliert werden, auch wenn die Map noch nicht vorhanden ist.

Im Programm definierte INPUT-Felder

Wenn Sie Namen der Eingabefelder (*operand1*) im Programm definieren, müssen diese nicht mit den für die Map verwendeten Feldnamen übereinstimmen.

Die Reihenfolge der Felder im Programm muß allerdings zur Reihenfolge der Felder in der Map passen. Hierbei ist zu beachten, daß der Map-Editor die in der Map definierten Felder in alphabetischer Reihenfolge der Feldnamen sortiert. Näheres hierzu finden Sie in der Map-Editor-Beschreibung in Ihrem *Natural Benutzerhandbuch* bzw. *User's Guide*.

Das Programm-Editor-Zeilenskommando “.I(*mapname*)” kann dazu verwendet werden, ein vollständiges INPUT USING MAP-Statement mit einer Parameterliste, die anhand der in der angegebenen Map definierten Felder generiert wird, zu erstellen.

Wird das Layout der Map verändert, muß das Programm nicht neu katalogisiert werden, es sei denn, in der Map werden Felder gelöscht, hinzugefügt oder umbenannt, Feldformate/-längen geändert oder Array-Strukturen modifiziert.

Bei der Ausführung prüft Natural, ob Format und Länge der Map-Felder in Einklang mit denen der Programm-Felder stehen. Ist dies nicht der Fall, wird eine entsprechende Fehlermeldung ausgegeben.

INPUT WINDOW=*'window-name'*

Diese Option ist unter **Syntax 1** des INPUT-Statements beschrieben.

WITH TEXT/MARK/ALARM-options

Diese Optionen sind unter **Syntax 1** des INPUT-Statements beschrieben.

USING MAP

Mit der USING MAP-Klausel wird eine Map-Definition aufgerufen, die vorher mit dem Map-Editor in einer Natural-Systemdatei gespeichert worden ist.

Der *map-name* kann als 1 bis 8 Zeichen lange alphanumerische Konstante oder in Form einer Benutzervariablen angegeben werden. Wird eine Variable verwendet, muß diese vorher definiert worden sein. Der Map-Name darf ein Und-Zeichen (&) enthalten; dies wird dann zur Ausführungszeit durch den aktuellen Inhalt der Natural-Systemvariablen *LANGUAGE ersetzt. Dadurch ist es möglich, verschiedensprachige Maps aufzurufen.

Die Ausführung des INPUT-Statements löscht den bisherigen Bildschirminhalt, es sei denn, Sie verwenden eine NO ERASE-Klausel (siehe unten), wobei dann die Map den aktuellen Inhalt des Bildschirms überlagert.

NO ERASE

Diese Option ist unter **Syntax 1** des INPUT-Statements beschrieben.

Felder (*operand1*)

Die zu verwendenden Felder. Es können Datenbankfelder oder Benutzervariablen angegeben werden, die jedoch alle vorher definiert worden sein müssen. Die Felder müssen in Anzahl, Reihenfolge Format und Länge mit den referenzierten Map-Feldern übereinstimmen, da andernfalls ein Fehler generiert wird.

Ändert sich der Wert eines Datenbankfeldes aufgrund einer INPUT-Verarbeitung, so betrifft dies nur den Feldwert im Datenbereich. Um den Wert eines Feldes auf der Datenbank zu ändern, sind entsprechende UPDATE-/STORE-Statements erforderlich.

INPUT-Statement unter Nicht-Screen-Modi

Forms-Modus

Der Forms-Modus wird mit dem Terminalkommando `%F` eingeschaltet.

Im Forms-Modus zeigt Natural den gesamten Ausgabertext des Map-Layouts Feld für Feld an, und zwar entsprechend der Positionierungsparameter. Dadurch kann der Benutzer Daten Feld für Feld eingeben. Wenn alle Daten eingegeben sind, wird eine Hardcopy-Ausgabe erzeugt, die genau dem Abbild auf dem Bildschirm entspricht.

Im Falle eines Fehlers kann der Benutzer das Terminalkommando `%R` eingeben und sodann das gesamte Formular erneut eingeben. Die Eingaben werden dann wie bei der ersten Ausführung des INPUT-Statements verarbeitet.

Keyword/Delimiter-Modus

Der Keyword/Delimiter-Modus wird mit dem Terminalkommando `%D` eingeschaltet.

In diesem Modus können Daten a) unter Verwendung von Schlüsselwörtern oder b) in Abhängigkeit von der Position/Reihenfolge der Felder eingegeben werden, wie auf der folgenden Seite beschrieben.

a) Der Text, der im Forms-Modus vor einem Feld ausgegeben würde, wird im Keyword/Delimiter-Modus als Schlüsselwort (Keyword) verwendet, um das betreffende Feld zu identifizieren. Nach dem Schlüsselwort muß das Input-Assign-Zeichen stehen (siehe Session-Parameter IA) und unmittelbar danach die Eingabedaten.

Die Eingabedaten werden durch das mit dem Session-Parameter ID definierte Input-Delimiterzeichen voneinander getrennt. Auf ein Input-Assign-Zeichen folgende Leerzeichen werden als Eingabedaten interpretiert. Nach dem letzten Datenelement ist kein Input-Delimiterzeichen erforderlich.

Die Reihenfolge der Eingabedaten ist beliebig. Wird ein nicht im INPUT-Statement definiertes Schlüsselwort eingegeben, gibt Natural eine entsprechende Fehlermeldung aus. Es brauchen nicht alle Felder mit Eingabedaten gefüllt zu werden. Felder, in die nichts eingegeben wird, werden auf Leerzeichen (alphanumerische Felder) bzw. Null (numerische und hexadezimale Felder) gesetzt.

b) Es können auch lediglich Eingabedaten, jeweils durch das gültige Input-Delimiter-Zeichen voneinander getrennt, eingegeben werden. In diesem Fall muß die Reihenfolge der Eingabedaten der Reihenfolge der Eingabefelder im INPUT-Statement entsprechen.

Beide Eingabearten (a und b) können auch miteinander kombiniert werden. Solange keine Schlüsselwörter angegeben werden, ist die im INPUT-Statement angegebene Reihenfolge der Felder maßgeblich; wird ein Schlüsselwort angegeben, so werden die auf das Schlüsselwort folgenden Eingabedaten dem durch das Schlüsselwort identifizierten Feld zugeordnet. Folgen hierauf wieder Eingabedaten ohne Schlüsselwörter, so werden diese Daten den Feldern zugeordnet, die im INPUT-Statement auf das mit Schlüsselwort identifizierte Feld folgen.

Anmerkung:

Ein Schlüsselwort und das dazugehörige Eingabefeld müssen sich auf derselben logischen Zeile befinden. Wenn die Länge beider zusammen die Zeilenlänge überschreitet, vergrößern Sie die Zeilenlänge (Session-Parameter LS) entsprechend, so daß Schlüsselwort und Feld in eine Zeile passen.

Im Keyword/Delimiter-Modus eingegebene Eingabedaten werden entsprechend der auch im Screen-Modus gültigen Regeln auf ihre Gültigkeit überprüft. Wird versucht, einen Wert einzugeben, der länger als das betreffende Feld ist, so gibt Natural eine entsprechende Fehlermeldung aus.

Werden Daten für ein INPUT-Statement im Keyword/Delimiter-Modus von einem gepufferten Terminal (Typ 3270) oder einem PC unter Windows aus eingegeben, so müssen alle von einem bestimmten INPUT-Statement zu verarbeitenden Eingabedaten auf einem einzigen Schirm eingegeben werden; erst wenn alle Daten eingegeben sind, darf FREIG gedrückt werden.

Input-Daten aus dem Natural-Stack

Daten, die mittels eines FETCH-, RUN- oder STACK-Statements im Natural-Stack abgelegt wurden, werden bei der Ausführung des nächsten INPUT-Statements als Eingabedaten verarbeitet.

Das INPUT-Statement verarbeitet die Daten in dem oben beschriebenen Keyword/Delimiter-Modus.

Felder, für die keine Eingabedaten zur Verfügung stehen, werden je nach Format mit Leerzeichen bzw. Nullen gefüllt. Sind mehr Daten als Eingabefelder vorhanden, so werden überschüssige Eingabedaten ignoriert.

Wenn ein Feld mit Daten aus dem Stack gefüllt wird, gelten die Feldattribute nicht für die Daten.

Über die Natural-Systemvariable *DATA können Sie erfahren, wieviele Datenelemente gegenwärtig im Natural-Stack zur Verfügung stehen.

INPUT-Statement im Batch-Betrieb auf Großrechnern

Forms-Modus

Im Batch-Forms-Modus wird die Eingabe-Map angezeigt. Für jede Zeile, die ein oder mehrere Eingabefelder (AD=A oder AD=M) enthält, wird ein Datensatz gelesen, und die in dem Satz enthaltenen Daten werden den entsprechenden Feldern zugeordnet.

Eingabedatenfelder werden als benachbart angesehen. Werden keine Delimiter-Zeichen verwendet, so müssen die Daten genau den intern definierten Feldlängen entsprechend eingegeben werden. Bei numerischen Feldern müssen gegebenenfalls eine Stelle für das Vorzeichen (falls SG=ON gesetzt ist) und/oder eine Stelle für das Komma (Dezimalpunkt) berücksichtigt werden.

Werden Delimiter-Zeichen verwendet, um die Eingabedaten der einzelnen Felder voneinander zu trennen, so ist dies nicht erforderlich; die Eingabedaten werden dann jeweils von links nach rechts ab Stelle 1 verarbeitet. Für die Eingabe der Daten gelten die unter **Eingabe von Daten als Reaktion auf ein INPUT-Statement** (Seite 376) beschriebenen Regeln. Darüber hinaus kann mit dem Assign-Zeichen bewirkt werden, daß der Inhalt eines *OUTIN-Feldes nicht zurückgesetzt wird.

Keyword/Delimiter-Modus

Im Batch-Betrieb gilt für diesen Modus dasselbe wie im TP-Betrieb, allerdings mit folgenden Ausnahmen:

- Das Drucken der gesamten Eingabe-Map kann über das Terminalkommando %Q gesteuert werden.
- *OUTIN-Felder behalten ihre ursprünglichen Werte, wenn diese nicht explizit geändert werden.

Terminalkommandos im Batch-Betrieb

Folgende Terminalkommandos können eingesetzt werden, wenn ein INPUT-Statement im Batch-Betrieb auf einem Großrechner verwendet wird:

Kommando	Funktion
%*	Wenn in Position 1 oder 2 eines Datensatzes eingegeben, bewirkt %* die Unterdrückung der Ausgabe des nächsten Eingabedatensatzes: DATA RECORD %* SUPPRESSED DATA RECORD
%	Als letztes Zeichen eines Datensatzes bewirkt es, daß der nachfolgende Eingabedatensatz als Fortsetzung des aktuellen Datensatzes interpretiert wird. DATA, RECORD, WITH, CONTINUATION, % CONTINUATION RECORD INPUT V1 V2 V3 V4 V5 V6 DISPLAY V1 V2 V3 V4 V5 V6 erzeugt folgende Ausgabe: DATA RECORD WITH CONTINUATION CONTINUATION RECORD
%/	Bewirkt eine "End-of-File"-Bedingung, wenn es in den ersten zwei Positionen eines Eingabedatensatzes (ohne nachfolgende Nicht-Leerzeichen) eingegeben wird.
% %	Restart-Punkt im Eingabedatenstrom setzen.
%.	Lesen der Eingabewerte für das gerade ausgeführte INPUT-Statement wird beendet.
% K <i>nn</i>	Simulieren von PF-Tasten.
% KP <i>n</i>	Simulieren von PA-Tasten.
% Q	Dieses Kommando bewirkt, daß Maps, die zum Lesen von Eingabedaten verwendet werden, nicht mit ausgedruckt werden.

Nähere Informationen über Terminalkommandos finden Sie im *Natural Referenzhandbuch*.

Zur Verwendung des INPUT-Statements im Batch-Betrieb ist zusätzliche JCL erforderlich. Bitte wenden Sie sich an Ihren Natural-Administrator um sicherzustellen, daß diese JCL bereitgestellt ist, bevor Sie eine Batch-Ausführung versuchen.

INTERFACE

```
INTERFACE interface-name  
  
    [ EXTERNAL ]  
    [ ID interface-GUID ]  
    [ property-definition ]  
    [ method-definition ]  
  
END-INTERFACE
```

Funktion

Ein Interface (Schnittstelle) ist eine Sammlung von Methoden und Eigenschaften, die semantisch zusammengehören und eine bestimmte Funktion einer Klasse darstellen.

Sie können ein oder mehrere Interfaces für eine Klasse definieren. Durch die Definition mehrerer Interfaces wird es Ihnen ermöglicht, Methoden nach ihrer Funktionsweise zu strukturieren/gruppieren, z.B. stellen Sie alle Methoden, die mit Dauerhaftigkeit (Laden, Speichern, Aktualisieren) zu tun haben, in ein Interface und die anderen Methoden in andere Interfaces.

Das INTERFACE-Statement dient zur Definition eines Interface. Es darf nur innerhalb eines Natural-Klassenmoduls verwendet werden und kann wie folgt definiert werden:

- innerhalb eines DEFINE CLASS-Statements. Diese Form wird verwendet, wenn das Interface nur in einer Klasse implementiert werden soll.

ODER

- in innerhalb der INTERFACE USING-Klausel des DEFINE CLASS-Statements enthaltenem Copycode. Diese Form wird verwendet, wenn das Interface in mehr als einer Klasse implementiert werden soll.

Die mit dem Interface verbundenen Eigenschaften und Methoden werden in den “Property Definitions” bzw. “Method Definitions” festgelegt.

interface-name

Dies ist der dem Interface zuzuweisende Name. Der Interface-Name kann maximal 32 Zeichen lang sein und muß den Natural-Namenskonventionen für Benutzervariablen entsprechen (weitere Informationen finden Sie im *Natural Referenzhandbuch* oder der *Natural Reference Documentation*). Er muß pro Klasse eindeutig und nicht mit dem Klassen-Namen identisch sein.

Soll das Interface von Clients verwendet werden, die in anderen Programmiersprachen geschrieben sind, sollte der Interface-Name so gewählt sein, daß er nicht gegen die für diese Sprachen geltenden Namenskonventionen verstößt. Bolero verwendet zum Beispiel die Namenskonventionen für Java. So sollte ein für einen Bolero-Client vorgesehenes Interface auch nicht gegen die Namenskonventionen von Java verstoßen.

EXTERNAL-Klausel

Anmerkung:

Diese Klausel steht nur für Natural Version 5.1.1 für UNIX/OpenVMS und Windows oder einer höheren Version zur Verfügung.

Die EXTERNAL-Klausel wird verwendet, um anzuzeigen, dass dieses Interface von der Klasse implementiert wird, die aber ursprünglich in einer unterschiedlichen Klasse definiert worden ist. Die Klasse ist nur relevant, wenn sie in DCOM registriert werden soll. Interfaces mit der EXTERNAL-Klausel werden ignoriert, wenn die Klasse in DCOM registriert wird. Es wird davon ausgegangen, dass das Interface von der sie ursprünglichen definierenden Klasse registriert wird.

ID-Klausel

Die ID-Klausel wird verwendet, um dem Interface eine weltweit gültige ID (“globally unique ID”) zuzuweisen. Die Interface-GUID ist der Name einer GUID, die in einer Data Area definiert ist, welche in der LOCAL-Klausel enthalten ist. Die Interface-GUID ist eine (benannte) Alpha-Konstante. Eine GUID muß einem Interface zugewiesen werden, wenn die Klasse in DCOM registriert werden soll.

property-definition

```
PROPERTY property-name  
    [ (format-length/array-definition) ]  
    [ ID dispatch-ID ]  
    [ READONLY ]  
    [ IS operand ]  
  
END-PROPERTY
```

Die Funktion “Property Definition” dient zur Definition von Eigenschaften für das Interface.

“Properties” sind Attribute eines Objekts, das von Clients aufgerufen werden kann. Ein Objekt, das einen Angestellten darstellt, kann zum Beispiel eine Property mit Namen ‘Name’ und eine andere mit Namen ‘Department’ haben. Das Einlesen oder Ändern des Namens oder der Abteilung des Angestellten durch Aufruf der Property für dessen Namen oder dessen Abteilung ist viel einfacher für einen Client als eine Methode aufzurufen, die den Wert zurückgibt, und eine andere Methode aufzurufen, die den Wert ändert.

Jede Property benötigt eine Variable in der Object Data Area der Klasse, um dessen Wert zu speichern – dies wird als Objektdaten-Variable bezeichnet. Die Property-Definition dient dazu, diese Variable für den Zugriff von Clients freizugeben. Die Property-Definition legt den Namen und das Format der Property fest und verbindet sie mit der Objektdaten-Variable. Im einfachsten Fall übernimmt die Property den Namen und das Format der Objektdaten-Variable selbst. Es ist auch möglich, den Namen und das Format innerhalb bestimmter Grenzen zu überschreiben.

property-name

Dies ist der der Property zuzuweisende Name. Der Property-Name kann maximal bis zu 32 Zeichen enthalten und muß den Natural-Namenskonventionen für Benutzervariablen entsprechen (weitere Informationen finden Sie im *Natural Referenzhandbuch oder der Natural Reference Documentation*).

Soll die Property von Clients verwendet werden, die in anderen Programmiersprachen geschrieben sind, sollte der Property-Name so gewählt sein, daß er nicht gegen die für diese Sprachen geltenden Namenskonventionen verstößt. Bolero verwendet zum Beispiel die Namenskonventionen für Java. So sollte eine für einen Bolero-Client vorgesehene Property auch nicht gegen die Namenskonventionen von Java verstoßen.

format-length/array-definition

Damit wird das Format der Property definiert, wie es von den Clients erkannt wird.

Wird *format-length/array-definition* weggelassen, wird *format-length* und *array-definition* aus der in der IS-Klausel zugewiesenen Objektdaten-Variable genommen.

Wenn *format-length/array-definition* angegeben wird, muß diese Angabe unbedingt datenübertragungskompatibel sein, und zwar zu dem und von dem in *operand* in der IS-Klausel angegebenen Format der Objektdaten-Variable. Im Falle einer READONLY-Property braucht die Datenübertragungs-Kompatibilität ausschließlich in einer Richtung zu gelten: mit der Objektdaten-Variable als Ausgangsoperand und der Property als Zieloperand.

Wenn eine Array-Definition angegeben wird, muß sie in den Dimensionen, Ausprägungen pro Dimension, Untergrenzen und Obergrenzen mit der Array-Definition der entsprechenden Objektdaten-Variable übereinstimmen. Dies kommt durch Spezifikation eines Sternchens (*) für jede Dimension zum Ausdruck.

ID-Klausel

Anmerkung:

Diese Klausel steht nur für Natural Version 5.1.1 für UNIX/OpenVMS und Windows oder einer höheren Version zur Verfügung.

Die ID-Klausel wird verwendet, um einer Property einen spezifischen numerischen Bezeichner (Identifizier) zuzuweisen. Dieser Identifizier (die sogenannte Dispatch-ID) ist nur relevant, wenn die Klasse in DCOM registriert werden soll.

Normalerweise weist Natural einer Property automatisch eine Dispatch-ID zu. Es ist nur erforderlich, explizit eine spezifische Dispatch-ID für eine Property zu definieren, wenn die Property zu einem Interface zur EXTERNAL-Klausel gehört. (Dies ist ein Interface, das in dieser Klasse implementiert werden soll, aber ursprünglich in einer unterschiedlichen Klasse definiert worden ist.) In diesem Fall werden die zu verwendenden Dispatch-IDs gewöhnlich von der ursprünglichen Implementierung des Interface vorgeschrieben.

Die Dispatch-ID ist eine positive Konstante ungleich Null mit Format I4.

READONLY

Wenn dieses Schlüsselwort angegeben wird, kann der Wert der Property nur gelesen und nicht gesetzt werden. Das in *operand* in der IS-Klausel angegebene Format der Objektdaten-Variable muß datenübertragungskompatibel mit dem in *format-length/array-definition*. angegebenen Format sein. Es muß nicht datenübertragungskompatibel in der umgekehrten Richtung sein.

Wenn das Schlüsselwort READONLY weggelassen wird, kann der Property-Wert sowohl gelesen als auch gesetzt werden.

IS-Klausel

Der *operand* in der IS-Klausel weist eine Objektdaten-Variable als Adresse zu, um den Property-Wert zu speichern. Die zugewiesene Objektdaten-Variable muß nicht unbedingt eine Gruppe sein. Die Variable wird in normaler Operanden-Syntax referenziert. Dies bedeutet, daß wenn die Objektdaten-Variable ein Array ist, sie mit Index-Notation referenziert werden muß. Nur die vollständige Indexbereichs-Notation und Sternchen-Notation ist zulässig.

Die IS-Klausel sollte nicht verwendet werden, wenn das INTERFACE-Statement aus einem Copycode-Member übernommen und in mehreren Klassen wiederverwendet wird. Möchten Sie das INTERFACE-Statement nochmals verwenden, müssen Sie die Objektdaten-Variable in einem PROPERTY-Statement außerhalb des INTERFACE-Statements zuweisen.

Wenn die IS-Klausel weggelassen wird, wird die Property mit der Objektdaten-Variable mit demselben Namen wie die Property verknüpft. Wenn eine Variable mit diesem Namen nicht definiert ist, oder wenn es sich um eine Gruppe handelt, führt dies zu einem Syntax-Fehler.

Beispiele

Angenommen die Object Data Area enthält die folgenden Daten-Definitionen:

```
1 Salary(p7.2)
  1 SalaryHistory(p7.2/1:10)
```

dann sind die folgenden Property-Definitionen erlaubt:

```
property Salary
end-property
property Pay is Salary
end-property
property Pay(P7.2) is Salary
end-property
property Pay(N7.2) is Salary
end-property
property SalaryHistory
end-property
property OldPay is SalaryHistory(*)
end-property
property OldPay is SalaryHistory(1:10)
end-property
property OldPay(P7.2/*) is SalaryHistory(1:10)
end-property
property OldPay(N7.2/*) is SalaryHistory(*)
end-property
```

Die folgenden Property-Definitionen sind nicht zulässig:

```
/* Not data transfer-compatible. */
property Pay(L) is Salary
end-property
/* Not data transfer-compatible. */
property OldPay(L/*) is SalaryHistory(*)
end-property
/* Not data transfer-compatible. */
property OldPay(L/1:10) is SalaryHistory(1:10)
end-property
/* Assigns an array to a scalar. */
property OldPay(P7.2) is SalaryHistory(1:10)
end-property
/* Takes only a sub-array. */
property OldPay(P7.2/3:5) is SalaryHistory(*)
end-property
/* Index specification omitted in ODA variable SalaryHistory. */
property OldPay is SalaryHistory
end-property
/* Only asterisk notation allowed in property format specification. */
property OldPay(P7.2/1:10) is SalaryHistory(*)
end-property
```

method-definition

METHOD	<i>method-name</i> [ID dispatch-id] [IS subprogram-name] [PARAMETER { USING { local-data-area parameter-data-area } }] ... data-definition...
END-METHOD	

Die “Method Definition” dient zur Definition einer Methode für das Interface.

Um das Interface in verschiedenen Klassen wiederverwenden zu können, übernehmen Sie die Interface-Definition aus einem Copycode und definieren Sie das Subprogramm hinter der Interface-Definition mit einem METHOD-Statement. Dann können Sie die Methode in verschiedenen Klassen anders implementieren.

method-name

Dies ist der der Methode zuzuweisende Name. Der Methoden-Name kann maximal bis zu 32 Zeichen enthalten und muß den Natural-Namenskonventionen für Benutzervariablen entsprechen (weitere Informationen entnehmen Sie dem *Natural Referenzhandbuch* oder der *Natural Reference Documentation*). Er muß pro Interface eindeutig sein.

Soll die Methode von Clients verwendet werden, die in anderen Programmiersprachen geschrieben sind, sollte der Methoden-Name so gewählt sein, daß er nicht gegen die für diese Sprachen geltenden Namenskonventionen verstößt. Bolero verwendet zum Beispiel die Namenskonventionen für Java. So sollte eine für einen Bolero-Client vorgesehene Methode auch nicht gegen die Namenskonventionen von Java verstoßen.

ID-Klausel

Anmerkung:

Diese Klausel steht nur für Natural Version 5.1.1 für UNIX/OpenVMS und Windows oder einer höheren Version zur Verfügung.

Die ID-Klausel wird verwendet, um einer Methode einen spezifischen numerischen Bezeichner (Identifizier) zuzuweisen. Dieser Identifizier (die sogenannte Dispatch-ID) ist nur relevant, wenn die Klasse in DCOM registriert werden soll.

Normalerweise weist Natural einer Methode automatisch eine Dispatch-ID zu. Es ist nur erforderlich, explizit eine spezifische Dispatch-ID für eine Property zu definieren, wenn die Property zu einem Interface zur EXTERNAL-Klausel gehört. (Dies ist ein Interface, das in dieser Klasse implementiert werden soll, aber ursprünglich in einer unterschiedlichen Klasse definiert worden ist.) In diesem Fall werden die zu verwendenden Dispatch-IDs gewöhnlich von der ursprünglichen Implementierung des Interface vorgeschrieben.

Die Dispatch-ID ist eine positive Konstante ungleich Null mit Format I4.

IS-Klausel

Dies ist der Name des die Methode implementierenden Subprogramms. Der Name des Subprogramms besteht aus bis zu 8 Zeichen. Die Voreinstellung ist *method-name* (wenn die IS-Klausel nicht angegeben wird).

PARAMETER-Klausel

Mit dieser Klausel werden die Parameter der Methode angegeben; sie hat dieselbe Syntax wie die PARAMETER-Klausel des DEFINE DATA-Statements. Weitere Informationen zum DEFINE DATA-Statement entnehmen Sie der *Natural Statements Dokumentation*.

Die Parameter müssen mit den Parametern übereinstimmen, die später bei der Implementierung des Subprogramms verwendet werden. Dies wird am besten durch Verwendung einer Parameter Data Area gewährleistet.

In der Parameter Data Area als 'BY VALUE' markierte Parameter sind Eingabe-Parameter der Methode.

Nicht als 'BY VALUE' markierte Parameter werden 'by reference' übergeben und sind Eingabe/Ausgabe-Parameter. Dies ist die Voreinstellung.

Der als 'BY VALUE RESULT' markierte erste Parameter wird als Rückgabewert für die Methode zurückgegeben. Wenn mehr als ein Parameter so markiert ist, werden die anderen als Eingabe/Ausgabe-Parameter behandelt.

Als 'OPTIONAL' markierte Parameter stehen mit Version 4.1.2 und allen nachfolgenden Releases zur Verfügung. Optionale Parameter müssen nicht angegeben werden, wenn die Methode aufgerufen wird. Auf ihre Angabe können Sie verzichten, indem Sie die Notation *nX* im SEND METHOD-Statement verwenden.

LIMIT

LIMIT n

Funktion

Das Statement LIMIT dient dazu, die Anzahl der Durchläufe einer Verarbeitungsschleife, die mit FIND, READ oder HISTOGRAM initiiert wurde, zu begrenzen.

Das festgesetzte Limit gilt für alle nachfolgenden Verarbeitungsschleifen des Programms, bis es durch ein weiteres LIMIT-Statement außer Kraft gesetzt wird. Das LIMIT-Statement gilt nicht für einzelne Statements, in denen ausdrücklich ein anderes Limit angegeben ist (z.B. FIND (n) ...).

Wenn das Limit erreicht ist, wird die Verarbeitung der betreffenden Schleife abgebrochen und eine entsprechende Meldung ausgegeben (vgl. Session-Parameter LE im *Natural Referenzhandbuch*).

Wird kein LIMIT-Statement verwendet, so gilt standardmäßig das bei der Natural-Installation festgesetzte Limit.

Angabe des Limits (n)

Das Limit n muß als numerische Konstante angegeben werden und kann einen Wert von 0 bis 99999999 (führende Nullen sind optional) haben. Ist das Limit auf 0 gesetzt, wird die Schleife nicht durchlaufen.

Zählweise

Um zu ermitteln, ob eine Verarbeitungsschleife das Limit erreicht hat, wird jeder mit der Schleife gelesene Datensatz gezählt; hierbei gilt:

- Ein Datensatz, der aufgrund der WHERE-Bedingung eines FIND- oder READ-Statements zurückgewiesen wird, wird *nicht* mitgezählt.
- Ein Datensatz, der aufgrund eines ACCEPT- oder REJECT-Statements zurückgewiesen wird, *wird* mitgezählt.

Beispiel 1

```

/* EXAMPLE 'LMTEX1': LIMIT
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 CITY
END-DEFINE
/*****
LIMIT 4
/*****
READ EMPLOY-VIEW BY NAME STARTING FROM 'BAKER'
  DISPLAY NOTITLE NAME PERSONNEL-ID CITY *COUNTER
END-READ
/*****
END

```

NAME	PERSONNEL ID	CITY	CNT
BAKER	20016700	OAK BROOK	1
BAKER	30008042	DERBY	2
BALBIN	60000110	BARCELONA	3
BALL	30021845	DERBY	4

Beispiel 2

```

/* EXAMPLE 'LMTEX2': LIMIT
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
END-DEFINE
/*****
LIMIT 3
/*****
FIND EMPLOY-VIEW WITH NAME > 'A'
  READ EMPLOY-VIEW BY NAME STARTING FROM 'BAKER'
    DISPLAY NOTITLE 'CNT(0100)' *COUNTER(0100)
                      'CNT(0110)' *COUNTER(0110)

  END-READ
END-FIND
/*****
END

```

CNT(0100)	CNT(0110)
1	1
1	2
1	3
2	1
2	2
2	3
3	1
3	2
3	3

LOOP

Anmerkung:

Dieses Statement darf nur im Reporting Mode verwendet werden; im Structured Mode ist es nicht erlaubt.

[CLOSE] LOOP [(r)]

Funktion

Das Statement LOOP dient dazu, eine Verarbeitungsschleife zu schließen. Es bewirkt, daß der aktuelle Schleifendurchlauf beendet wird und die Kontrolle wieder an den Anfang der Schleife übergeben wird.

Sobald die Verarbeitungsschleife, auf die sich das LOOP-Statement bezieht, beendet ist (d.h. sobald alle Datensätze verarbeitet und alle Schleifendurchläufe ausgeführt sind), wird die Verarbeitung mit dem auf das LOOP-Statement folgenden Statement fortgesetzt.

Statement-Referenzierung (r)

Sollen mehrere Schleifen geschlossen werden, so kann ein bestimmtes Statement per Statement-Label oder Sourcecode-Zeilenummer referenziert werden; in diesem Falle werden durch das LOOP-Statement dann die referenzierte Verarbeitungsschleife sowie alle innerhalb der referenzierten Schleife befindlichen Schleifen geschlossen. Ist nichts anderes angegeben, bezieht sich das LOOP-Statement auf die innerste aktive Schleife.

Referenzierung von Datenbankvariablen

Neben dem Schließen der Schleife(n) bewirkt das LOOP-Statement, daß alle Referenzierungen von Feldern, die in FIND-, FIND FIRST-, FIND UNIQUE-, READ- oder GET-Statements innerhalb der geschlossenen Schleife(n) verwendet werden, eliminiert werden.

Ein Feld, das in einem View enthalten ist, kann auch außerhalb einer mit LOOP geschlossenen Schleife referenziert werden, und zwar indem bei der Referenzierung der View-Name angegeben wird.

Einschränkung

Ein LOOP-Statement darf nicht an eine logische Bedingung wie etwa ein IF- oder AT BREAK-Statement geknüpft werden.

Beispiel 1

```
0010 FIND ...
0020   READ ...
0030   READ ...
0040 LOOP (0010)   /* schließt alle Schleifen
```

Beispiel 2

```
0010 FIND ...
0020   READ ...
0030   READ ...
0040   LOOP       /* schließt die in Zeile 0030 initiierte Schleife
0050   LOOP       /* schließt die in Zeile 0020 initiierte Schleife
0060 LOOP         /* schließt die in Zeile 0010 initiierte Schleife
```

METHOD

```
METHOD method-name  
  
      OF [ INTERFACE ] interface-name  
      IS subprogram-name  
  
END-METHOD
```

Funktion

Das METHOD-Statement weist ein Subprogramm als Implementierung zu einer Methode zu, und zwar außerhalb einer Interface-Definition. Es wird verwendet, wenn die betreffende Interface-Definition aus einem Copycode übernommen wird und auf eine klassenspezifische Weise implementiert werden soll.

Das METHOD-Statement kann nur innerhalb des DEFINE CLASS-Statements und im Anschluss an die Interface-Definition verwendet werden. Die angegebenen Interface- und Methoden-Namen müssen innerhalb der Interface-Definitionen festgelegt werden.

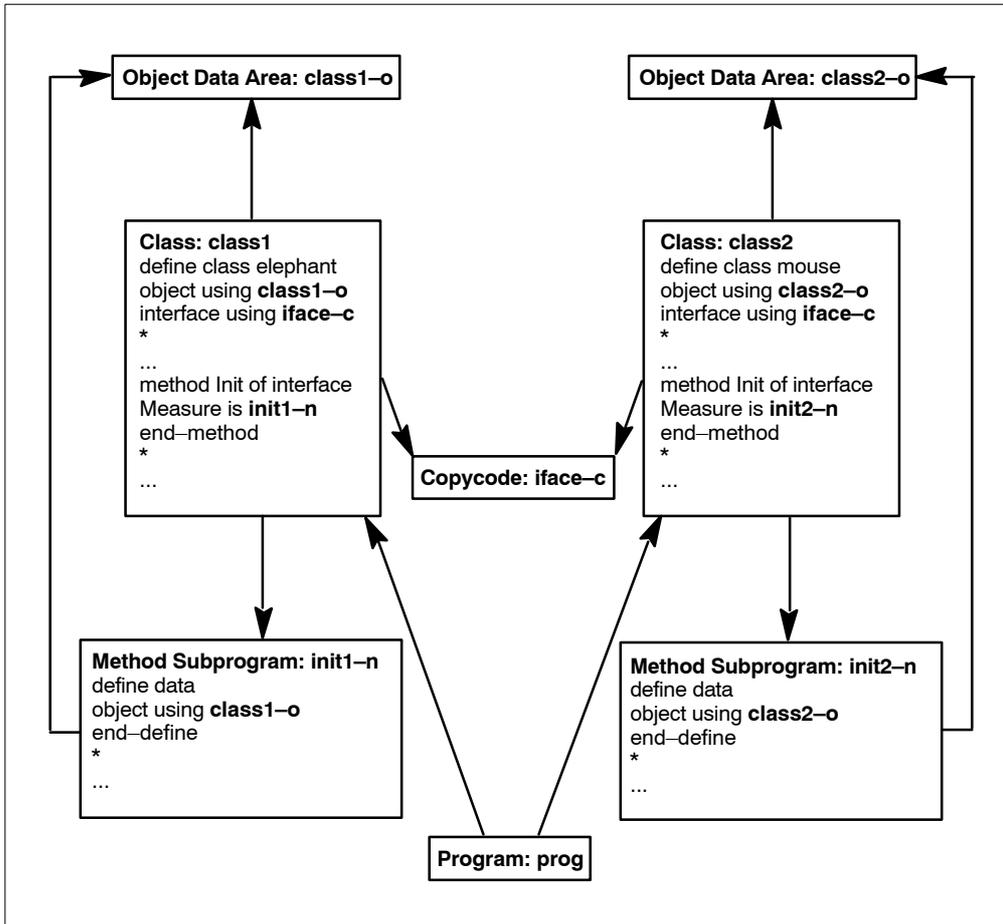
Beispiel:

Das folgende Beispiel zeigt, wie dasselbe Interface in zwei Klassen unterschiedlich implementiert wird und wie das PROPERTY-Statement und das METHOD-Statement zu diesem Zweck verwendet werden.

Die Interface *Measure* wird im Copycode *iface-c* definiert. Die Klassen “Elephant” und “Mouse” implementieren beide das Interface “Measure”. Deshalb beinhalten sie beide den Copycode *iface-c*.

Die Klassen implementieren aber die Property “Height” mittels verschiedener Variablen von ihren betreffenden Object Data Areas, und sie implementieren die Methode “Init” mit unterschiedlichen Subprogrammen. Sie verwenden das Statement PROPERTY, um die ausgewählte Datenbereichsvariable der Property zuzuweisen, und das Statement METHOD, um das ausgewählte Subprogramm der Methode zuzuweisen.

Jetzt kann das Programm “prog” Objekte beider Klassen erstellen und sie mittels derselben Methode “Init” initialisieren, wobei die Schritte der Initialisierung der betreffenden Klassen-Implementierung überlassen werden.



Im folgenden finden Sie den vollständigen Inhalt der im vorstehenden Beispiel verwendeten Natural-Module:

Copycode: iface-c

```

interface Measure
*
  property Height(p5.2)
  end-property
*
  property Weight(i4)
  end-property
*
  method Init
  end-method
*
end-interface

```

Class: class1

```

define class elephant
  object using class1-o
  interface using iface-c
*
  property Height of interface Measure is height
  end-property
*
  property Weight of interface Measure is weight
  end-property
*
  method Init of interface Measure is init1-n
  end-method
*
end-class
end

```

Object Data Area: class1-o

```

*   *** Top of Data Area ***
  1 HEIGHT                P 5.2
  1 WEIGHT                 I 2
*   *** End of Data Area ***

```

Method Subprogram: init1-n

```

define data
  object using class1-o
  end-define
*
  height := 17.3
  weight := 120
*
end

```

Class: class2

```
define class mouse
  object using class2-o
  interface using iface-c
  *
  property Height of interface Measure is size
  end-property
  *
  property Weight of interface Measure is weight
  end-property
  *
  method Init of interface Measure is init2-n
  end-method
  *
  end-class
end
```

Object Data Area: class2-o

```
*   *** Top of Data Area ***
1 SIZE                P 3.2
1 WEIGHT              I 1
*   *** End of Data Area ***
```

Method Subprogram: init2-n

```
define data
  object using class2-o
  end-define
  *
  size := 1.24
  weight := 2
  *end
```

Program: prog

```
define data local
  1 #o handle of object
  1 #height(p5.2)
  1 #weight(i4)
end-define
*
create object #o of class "Elephant"
send "Init" to #o
#height := #o.Height
#weight := #o.Weight
write #height #weight
*
create object #o of class "Mouse"
send "Init" to #o
#height := #o.Height
#weight := #o.Weight
write #height #weight
*
end
```

MOVE

MOVE [ROUNDED] operand1 [(parameter)] TO operand2...

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S A N	A N P I F B D T L C G O	ja	nein
Operand2	S A M	A N P I F B D T L C G O	ja	ja

**MOVE { SUBSTRING (^{operand1} operand1, operand3, operand4) } [(parameter)] TO
 { SUBSTRING (^{operand2} operand2, operand5, operand6) } ...**

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S A	A N*	ja	nein
Operand2	S A	A	ja	nein
Operand3	C S	N P I	ja	nein
Operand4	C S	N P I	ja	nein
Operand5	C S	N P I	ja	nein
Operand6	C S	N P I	ja	nein

* siehe Text.

MOVE BY { [NAME]
POSITION } operand1 TO operand2

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	G		ja	nein
Operand2	G		ja	nein

MOVE EDITED operand1 TO operand2 (EM=value)

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S A	A B	ja	nein
Operand2	S A	A N P I F B D T L	ja	ja

MOVE EDITED operand1 (EM=value) TO operand2

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S A N	A N P I F B D T L	ja	nein
Operand2	S A	A B	ja	ja

MOVE { LEFT
RIGHT } [JUSTIFIED] operand1 [(parameter)] TO operand2

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S A N	A N P I F B D T L	ja	nein
Operand2	S A	A	ja	ja

Verwandtes Statement

COMPUTE.

Funktion

Das Statement MOVE dient dazu, den Wert eines Operanden in einen oder mehrere andere Operanden (Felder oder Arrays) zu übertragen.

Ist *operand2* eine DYNAMISCHE Variable, kann ihre Länge mit der MOVE-Operation geändert werden. Die aktuelle Länge einer DYNAMISCHEN Variable kann mittels der Systemvariable *LENGTH bestimmt werden. Allgemeine Informationen zur dynamischen Variable DYNAMIC siehe *Natural Benutzerhandbuch* oder *Natural User's Guide*.

ROUNDED

Das Schlüsselwort ROUNDED bewirkt, daß *operand2* auf- bzw. abgerundet wird.

ROUNDED wird ignoriert, wenn *operand2* nicht numerisch ist.

Wenn *operand2* das Format N oder P hat und mehr als einmal angegeben wird, wird ROUNDED bei Zieloperanden mit 7 Stellen hinter dem Dezimalpunkt (Komma) ignoriert.

parameter

Als *parameter* können Sie die Option "PM=I" oder den Session-Parameter DF angeben:

PM=I

Zur Unterstützung von Sprachen, deren Schreibrichtung von rechts nach links ist, können Sie die Option “PM=I” angeben, um den Wert von *operand1* invers (d.h. von rechts nach links) in *operand2* zu übertragen.

Zum Beispiel hätte als Ergebnis der folgenden Statements das Feld #B den Inhalt “ZYX”:

```
MOVE 'XYZ' TO #A  
MOVE #A (PM=I) TO #B
```

PM=I kann nur angegeben werden, wenn *operand2* alphanumerisches Format hat.

Nachfolgende Leerzeichen in *operand1* werden entfernt, dann wird der Wert umgedreht und anschließend in *operand2* übertragen. Falls *operand1* nicht alphanumerisches Format hat, wird der Wert in alphanumerisches Format umgesetzt, bevor er umgedreht wird.

Zur Verwendung von PM=I zusammen mit MOVE LEFT/RIGHT JUSTIFIED, siehe Seite 428.

DF

Wenn *operand1* eine Datumsvariable und *operand2* ein alphanumerisches Feld ist, können Sie den Session-Parameter DF als *parameter* für diese Datumsvariable angeben. Der Session-Parameter DF ist im *Natural Referenzhandbuch* beschrieben.

SUBSTRING

Ohne SUBSTRING-Option wird der ganze Inhalt des Feldes übertragen. Die Option SUBSTRING ermöglicht es Ihnen, nur einen bestimmten Teil eines alphanumerischen Feldes zu übertragen. In der SUBSTRING-Klausel geben Sie nach dem Feldnamen (*operand1*) zunächst die erste Stelle (*operand3*) und dann die Länge (*operand4*) des Feldteils an, der übertragen werden soll.

Um zum Beispiel die 5. bis einschließlich 12. Stelle eines Feldes #A in ein Feld #B zu übertragen, würden Sie folgendes angeben:

```
MOVE SUBSTRING(#A,5,8) TO #B
```

Wenn *operand1* eine DYNAMISCHE Variable ist, muß der angegebene und zu übertragene Feldteil im Bereich seiner aktuellen Länge sein; sonst tritt ein Laufzeit-Fehler auf.

Sie können einen Wert eines alphanumerischen oder numerischen Feldes auch in einen bestimmten Teil des Zielfeldes übertragen. In der SUBSTRING-Klausel geben Sie nach dem Feldnamen (*operand2*) zunächst die erste Stelle (*operand5*) und dann die Länge (*operand6*) des Feldteils an, in den der Wert übertragen werden soll.

Um zum Beispiel den Wert eines Feldes #A in die 3. bis einschließlich 6. Stelle eines Feldes #B zu übertragen, würden Sie folgendes angeben:

```
MOVE #A TO SUBSTRING(#B,3,4)
```

Wenn *operand2* eine DYNAMISCHE Variable ist, darf die erste Stelle (*operand5*) nicht größer sein als die aktuelle Länge der Variable plus 1; eine höhere erste Stelle würde einen Laufzeit-Fehler zur Folge haben, weil dies zu einer nicht definierten Lücke im Inhalt von *operand2* führen würde.

Wenn Sie *operand3/5* weglassen, wird ab Anfang des Feldes übertragen. Wenn Sie *operand4/6* weglassen, wird ab der angegebenen Stelle (*operand3/5*) bis zum Ende des Feldes übertragen.

Wenn *operand2* eine DYNAMISCHE Variable ist und die erste Stelle (*operand5*) der aktuellen Länge der Variable plus 1 entspricht, was bedeutet, daß die MOVE-Operation verwendet wird, um die Länge der Variablen zu vergrößern, muß *operand6* angegeben werden, um die neue Länge der Variablen zu bestimmen.

Anmerkungen:

MOVE mit SUBSTRING-Option ist eine Byte-für-Byte-Übertragung (d.h. die unter "Arithmetische Operationen" im Natural Referenzhandbuch beschriebenen Regeln gelten hierbei nicht).

MOVE BY NAME

Mit dieser Option können Sie einzelne in einer Datenstruktur enthaltene Felder in eine andere Datenstruktur übertragen, und zwar unabhängig von ihrer Position innerhalb der Struktur. Ein Feld kann nur übertragen werden, wenn sein Name in beiden Datenstrukturen vorkommt (dies gilt auch für REDEFINE-ierte Felder sowie Felder, die aus einer Redefinition resultieren). Die einzelnen Felder können jedes beliebige Format haben. Die beiden Operanden können auch Views sein.

Anmerkung:

Die Reihenfolge der einzelnen Übertragungen ergibt sich aus der Reihenfolge der Felder in operand1.

MOVE BY NAME mit Arrays

Enthalten die Datenstrukturen Arrays, so werden diese bei der Übertragung intern mit Index “(*)” versehen; dies kann zu einem Fehler führen, falls die Arrays nicht den Zuweisungsbedingungen für Arrays (siehe Abschnitt **Verarbeitung von Arrays** im *Natural Referenzhandbuch*) entsprechen.

Beispiel 1 für MOVE BY NAME mit Arrays:

```
DEFINE DATA LOCAL
  1 #GROUP1
    2 #FIELD (A10/1:10)
  1 #GROUP2
    2 #FIELD (A10/1:10)
END-DEFINE
...
MOVE BY NAME #GROUP1 TO #GROUP2
...
```

In diesem Beispiel würde das MOVE-Statement intern wie folgt aufgelöst:

```
MOVE #GROUP1.#FIELD (*) TO #GROUP2.#FIELD (*)
```

Wenn ein Teil einer indizierten Gruppe in einen anderen Teil derselben Gruppe übertragen wird, kann dies zu unerwarteten Ergebnissen führen, wie im folgenden Beispiel veranschaulicht.

Beispiel 2 für MOVE BY NAME mit Arrays:

```
DEFINE DATA LOCAL
  1 #GROUP1 (1:5)
  2 #FIELDA (N1) INIT <1,2,3,4,5>
  2 REDEFINE #FIELDA
  3 #FIELDDB (N1)
END-DEFINE
...
MOVE BY NAME #GROUP1 (2:4) TO #GROUP1 (1:3)
...
```

In diesem Beispiel würde das MOVE-Statement intern wie folgt aufgelöst:

```
MOVE #FIELDA (2:4) TO #FIELDA (1:3)
MOVE #FIELDDB (2:4) TO #FIELDDB (1:3)
```

Zunächst wird der Inhalt der Ausprägungen 2 bis 4 von #FIELDA in die Ausprägungen 1 bis 3 von #FIELDA übertragen; d.h. die Ausprägungen erhalten folgende Werte:

Ausprägung:	1.	2.	3.	4.	5.
Wert vorher:	1	2	3	4	5
Wert nachher:	2	3	4	4	5

Dann wird der Inhalt der Ausprägungen 2 bis 4 von #FIELDDB in die Ausprägungen 1 bis 3 von #FIELDDB übertragen; d.h. die Ausprägungen erhalten folgende Werte:

Ausprägung:	1.	2.	3.	4.	5.
Wert vorher:	2	3	4	4	5
Wert nachher:	3	4	4	4	5

MOVE BY POSITION

Mit dieser Option können Sie Werte von Feldern einer Gruppe in Felder einer anderen Gruppe übertragen, und zwar unabhängig von den Namen der Felder. Die Werte werden Feld für Feld von einer Gruppe in die andere übertragen, und zwar in der Reihenfolge, in der die Felder definiert sind (Felder, die aus einer Redefinition resultieren, werden dabei nicht berücksichtigt).

Die einzelnen Felder können jedes beliebige Format haben. Die Anzahl der Felder in beiden Gruppen muß gleich sein; auch die Level-Struktur und die Array-Dimensionen der Felder müssen einander entsprechen. Format-Umsetzungen erfolgen entsprechend der im *Natural Referenzhandbuch* beschriebenen Regeln für arithmetische Operationen. Die beiden Operanden können auch Views sein.

Beispiel für MOVE BY POSITION:

```

DEFINE DATA LOCAL
  1 #GROUP1
    2 #FIELD1A (N5)
    2 #FIELD1B (A3/1:3)
    2 REDEFINE #FIELD1B
      3 #FIELD1BR (A9)
  1 #GROUP2
    2 #FIELD2A (N5)
    2 #FIELD2B (A3/1:3)
    2 REDEFINE #FIELD2B
      3 #FIELD2BR (A9)
END-DEFINE
...
MOVE BY POSITION #GROUP1 TO #GROUP2
...

```

In diesem Beispiel wird der Inhalt von #FIELD1A in #FIELD2A übertragen wird, und der Inhalt von #FIELD1B in #FIELD2B; die Felder #FIELD1BR und #FIELD2BR sind davon nicht betroffen.

MOVE EDITED

Für *operand1* oder *operand2* kann eine Editiermaske definiert werden.

Ist für *operand2* eine Editiermaske definiert, so wird der Wert von *operand1* unter Verwendung dieser Editiermaske in das Feld *operand2* übertragen.

Ist für *operand1* eine Editiermaske definiert, wird diese Editiermaske auf *operand1* angewandt und der Wert anschließend in *operand2* übertragen. Die Länge von *operand1* nach Anwendung der Editiermaske darf die Länge von *operand2* nicht überschreiten.

Einzelheiten zu Editiermasken finden Sie unter dem Session-Parameter EM im *Natural Referenzhandbuch*.

MOVE LEFT/RIGHT JUSTIFIED

Mit dieser Option können Sie festlegen, ob der übertragene Wert in *operand2* links- oder rechtsbündig ausgerichtet werden soll.

Bei MOVE LEFT JUSTIFIED werden vorangestellte Leerzeichen in *operand1* entfernt, bevor der Wert linksbündig in *operand2* übertragen wird. Der Rest von *operand2* wird dann gegebenenfalls mit Leerzeichen aufgefüllt. Falls der Wert länger als *operand2* ist, wird der Wert rechts abgeschnitten.

Bei MOVE RIGHT JUSTIFIED werden nachfolgende Leerzeichen in *operand1* entfernt, bevor der Wert rechtsbündig in *operand2* übertragen wird. Der Rest von *operand2* wird dann gegebenenfalls mit Leerzeichen aufgefüllt. Falls der Wert länger als *operand2* ist, wird der Wert links abgeschnitten.

MOVE LEFT/RIGHT JUSTIFIED kann nicht verwendet werden, wenn *operand2* eine DYNAMISCHE Variable ist.

MOVE LEFT/RIGHT JUSTIFIED mit PM=I

Wenn Sie MOVE LEFT/RIGHT JUSTIFIED in Verbindung mit PM=I verwenden, erfolgt die Übertragung in folgenden Schritten:

- ① Falls *operand1* nicht alphanumerisches Format hat, wird der Wert in alphanumerisches Format umgesetzt.
- ② Nachfolgende Leerzeichen in *operand1* werden entfernt.
- ③ Bei LEFT JUSTIFIED werden außerdem vorangestellte Leerzeichen in *operand1* entfernt.
- ④ Der Wert wird umgedreht und dann in *operand2* übertragen.
- ⑤ Gegebenenfalls wird *operand2* mit Leerzeichen aufgefüllt oder der Wert abgeschnitten (vgl. oben).

Weitere Hinweise

Wird ein Datenbankfeld als Ergebnisfeld (*operand2*) verwendet, so ändert sich der Wert des Feldes dadurch nur programmintern. Der in der Datenbank gespeicherte Feldwert wird davon nicht beeinflusst.

Eine Natural-Systemfunktion darf nur eingesetzt werden, wenn das MOVE-Statement in Verbindung mit einem AT BREAK-, AT END OF DATA- oder AT END OF PAGE-Statement verwendet wird.

Vgl. Abschnitt **Arithmetische Operationen** im *Natural Referenzhandbuch*.

Anmerkung:

Wenn operand1 eine Zeitvariable (Format T) ist, wird nur die Zeitkomponente des Variableninhalts übertragen, aber nicht die Datumskomponente (außer bei MOVE EDITED).

Beispiel 1

```

/* EXAMPLE 'MOVEX1': MOVE
/*****
DEFINE DATA LOCAL
1 #A (N3)
1 #B (A5)
1 #C (A2)
1 #D (A7)
1 #E (N1.0)
1 #F (A5)
1 #G (N3.2)
1 #H (A6)
END-DEFINE
/*****
MOVE 5 TO #A
WRITE NOTITLE 'MOVE 5 TO #A' 30X '=' #A
/*****
MOVE 'ABCDE' TO #B #C #D
WRITE 'MOVE ABCDE TO #B #C #D' 20X '=' #B '=' #C '=' #D
/*****
MOVE -1 TO #E
WRITE 'MOVE -1 TO #E' 28X '=' #E
/*****
MOVE ROUNDED 1.995 TO #E
WRITE 'MOVE ROUNDED 1.995 TO #E' 18X '=' #E
/*****
MOVE RIGHT JUSTIFIED 'ABC' TO #F
WRITE 'MOVE RIGHT JUSTIFIED 'ABC'' TO #F' 10X '=' #F
/*****
MOVE EDITED '003.45' TO #G (EM=999.99)
WRITE 'MOVE EDITED '003.45'' TO #G (EM=999.99)' 4X '=' #G
/*****
MOVE EDITED 123.45 (EM=999.99) TO #H
WRITE 'MOVE EDITED 123.45 (EM=999.99) TO #H' 6X '=' #H
/*****
END

```

MOVE 5 TO #A	#A: 5
MOVE ABCDE TO #B #C #D	#B: ABCDE #C: AB #D: ABCDE
MOVE -1 TO #E	#E: -1
MOVE ROUNDED 1.995 TO #E	#E: 2
MOVE RIGHT JUSTIFIED 'ABC' TO #F	#F: ABC
MOVE EDITED '003.45' TO #G (EM=999.99)	#G: 3.45
MOVE EDITED 123.45 (EM=999.99) TO #H	#H: 123.45

Beispiel 2

```

/* EXAMPLE 'MOVEX2': MOVE BY NAME
/*****
DEFINE DATA LOCAL
1 #SBLOCK
  2 #FIELD1 (A10) INIT <'AAAAAAAAAA'>
  2 #FIELD2 (A10) INIT <'BBBBBBBBBB'>
  2 #FIELD3 (A10) INIT <'CCCCCCCCCC'>
  2 #FIELD4 (A10) INIT <'DDDDDDDDDD'>
1 #TBLOCK
  2 #FIELD1 (A15) INIT <' '>
  2 #FIELD2 (A10) INIT <' '>
  2 #FIELD3 (A10) INIT <' '>
  2 #FIELD4 (A10) INIT <' '>
  2 #FIELD5 (A20) INIT <' '>
  2 #FIELD6 (A10) INIT <' '>
END-DEFINE
/*****
MOVE BY NAME #SBLOCK TO #TBLOCK
/*****
WRITE NOTITLE 'CONTENTS OF #TBLOCK AFTER MOVE BY NAME:'
  // '=' #TBLOCK.#FIELD1
  / '=' #TBLOCK.#FIELD2
  / '=' #TBLOCK.#FIELD3
  / '=' #TBLOCK.#FIELD4
  / '=' #TBLOCK.#FIELD5
  / '=' #TBLOCK.#FIELD6
/*****
END

```

CONTENTS OF #TBLOCK AFTER MOVE BY NAME:

```

#FIELD1:
#FIELD2: AAAAAAAAAA
#FIELD3:
#FIELD4: BBBBBBBBBB
#FIELD5:
#FIELD6: CCCCCCCC

```

MOVE ALL

MOVE ALL *operand1* **TO** *operand2* [**UNTIL** *operand3*]

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A N B	ja	nein
Operand2	S A	A	ja	ja
Operand3	C S	N P I	ja	nein

Funktion

Das Statement MOVE ALL dient dazu, den Wert von *operand1* so oft in *operand2* zu übertragen, bis letzterer voll ist.

Ausgangsoperand (*operand1*)

Dieser Operand enthält den zu übertragenden Wert.

Bei einem numerischen Operanden werden alle Stellen einschließlich vorangestellter Nullen übertragen.

Zieloperand (*operand2*)

In diesen Operanden wird der Wert übertragen. Der vorherige Inhalt des Operanden wird vor der MOVE ALL-Operation *nicht* zurückgesetzt. Dies ist insbesondere bei Verwendung der UNTIL-Option zu beachten, da bereits in *operand2* enthaltene Daten dort behalten werden, es sei denn, sie werden während der MOVE ALL-Operation ausdrücklich überlagert.

UNTIL-Option (*operand3*)

Mit der UNTIL-Option können Sie die MOVE ALL-Operation auf eine bestimmte Anzahl von Stellen in *operand2* begrenzen. Mit *operand3* geben Sie an, wieviele Stellen in *operand2* gefüllt werden sollen; ist diese Anzahl erreicht, wird die MOVE ALL-Operation beendet.

Ist *operand3* größer als *operand2* lang ist, wird die MOVE ALL-Operation beendet, sobald *operand2* vollständig gefüllt ist.

Die Option UNTIL kann auch verwendet werden, um einer DYNAMISCHEN Variable einen Startwert zuzuweisen: wenn *operand2* eine DYNAMISCHE Variable ist, entspricht ihre Länge nach der MOVE ALL-Operation dem Wert von *operand3*. Die aktuelle Länge einer DYNAMISCHEN Variable kann unter Verwendung der Systemvariable *LENGTH festgestellt werden. Allgemeine Informationen zu DYNAMISCHEN Variablen finden Sie im *Natural Benutzerhandbuch*.

Beispiel

```

/* EXAMPLE 'MOAEX1': MOVE ALL
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 FIRST-NAME
  2 NAME
  2 CITY
1 VEH-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
/*****
LIMIT 4
RD. READ EMPLOY-VIEW BY NAME
  SUSPEND IDENTICAL SUPPRESS
/*****
FD. FIND VEH-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (RD.)
  IF NO RECORDS FOUND
    MOVE ALL '*' TO FIRST-NAME (RD.)
    MOVE ALL '*' TO CITY (RD.)
    MOVE ALL '*' TO MAKE (FD.)
  END-NOREC
/*****
  DISPLAY NOTITLE (ES=OFF IS=ON ZP=ON AL=15)
    NAME (RD.) FIRST-NAME (RD.) CITY (RD.)
    MAKE (FD.) (IS=OFF)
/*****
  END-FIND
  END-READ
END

```

NAME	FIRST-NAME	CITY	MAKE
ABELLAN	*****	*****	*****
ACHIESON	ROBERT	DERBY	FORD
ADAM	*****	*****	*****
ADKINSON	TIMMIE	BEDFORD	GENERAL MOTORS

Äquivalentes Reporting-Mode-Beispiel: siehe Programm MOAEX1R in Library SYSEXRM.

MULTIPLY

Syntax 1

MULTIPLY [ROUNDED] *operand1* **BY** *operand2*

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S A M	N P I F	ja	nein
Operand2	C S A N	N P I F	ja	nein

Syntax 2

MULTIPLY [ROUNDED] *operand1* **BY** *operand2* **GIVING** *operand3*

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S A M	N P I F	ja	nein
Operand2	C S A N	N P I F	ja	nein
Operand3	S A M	A N P I F B	ja	ja

Verwandtes Statement

COMPUTE.

Funktion

Das Statement MULTIPLY dient dazu, zwei Operanden miteinander zu multiplizieren.

Zu Multiplikationen mit Arrays siehe auch den Abschnitt **Arithmetische Operationen mit Arrays** im *Natural Referenzhandbuch*.

Ergebnisfeld

Das Ergebnis der Multiplikation kann entweder in *operand1* oder in *operand3* ausgegeben werden.

Wird Syntax 1 verwendet, so wird das Ergebnis in *operand1* gespeichert.

Wird Syntax 2 (mit GIVING-Klausel) verwendet, bleibt *operand1* unverändert und das Ergebnis wird in *operand3* gespeichert.

Ist *operand1* eine numerische Konstante, dann muß eine GIVING-Klausel benutzt werden.

Wird ein Datenbankfeld als Ergebnisfeld verwendet, so ändert sich durch die Multiplikation nur der programmintern benutzte Wert des Feldes. Der in der Datenbank gespeicherte Feldwert wird davon nicht beeinflusst.

Beispiel

```

/* EXAMPLE 'MULEX1': MULTIPLY
/*****
DEFINE DATA LOCAL
1 #A (N3) INIT <20>
1 #B (N5)
1 #C (N3.1)
1 #D (N2)
1 #ARRAY1 (N5/1:4,1:4) INIT (2,*) <5>
1 #ARRAY2 (N5/1:4,1:4) INIT (4,*) <10>
END-DEFINE
/*****
MULTIPLY #A BY 3
WRITE NOTITLE 'MULTIPLY #A BY 3' 25X '=' #A
/*****
MULTIPLY #A BY 3 GIVING #B
WRITE 'MULTIPLY #A BY 3 GIVING #B' 15X '=' #B
/*****
MULTIPLY ROUNDED 3 BY 3.5 GIVING #C
WRITE 'MULTIPLY ROUNDED 3 BY 3.5 GIVING #C' 6X '=' #C
/*****
MULTIPLY 3 BY -4 GIVING #D
WRITE 'MULTIPLY 3 BY -4 GIVING #D' 15X '=' #D
/*****
MULTIPLY -3 BY -4 GIVING #D
WRITE 'MULTIPLY -3 BY -4 GIVING #D' 14X '=' #D
/*****
MULTIPLY 3 BY 0 GIVING #D
WRITE 'MULTIPLY 3 BY 0 GIVING #D' 14X '=' #D
/*****
WRITE / '=' #ARRAY1 (2,*) '=' #ARRAY2 (4,*)
MULTIPLY #ARRAY1 (2,*) BY #ARRAY2 (4,*)
WRITE 'MULTIPLY #ARRAY1 (2,*) BY #ARRAY2 (4,*)'
/ '=' #ARRAY1 (2,*) '=' #ARRAY2 (4,*)
/*****
END

```

MULTIPLY #A BY 3	#A:	60							
MULTIPLY #A BY 3 GIVING #B	#B:	180							
MULTIPLY ROUNDED 3 BY 3.5 GIVING #C	#C:	10.5							
MULTIPLY 3 BY -4 GIVING #D	#D:	-12							
MULTIPLY -3 BY -4 GIVING #D	#D:	12							
MULTIPLY 3 BY 0 GIVING #D	#D:	0							
#ARRAY1:	5	5	5	5	#ARRAY2:	10	10	10	10
MULTIPLY #ARRAY1 (2,*) BY #ARRAY2 (4,*)									
#ARRAY1:	50	50	50	50	#ARRAY2:	10	10	10	10

NEWPAGE

$\text{NEWPAGE } [(rep)] \left[\begin{array}{l} \text{EVEN [IF] TOP [OF] [PAGE]} \\ \left[\begin{array}{l} \text{IF} \\ \text{WHEN} \end{array} \right] \text{ LESS [THAN] } operand1 \text{ [LINES] [LEFT]} \\ \text{[WITH] TITLE} \end{array} \right] \quad (see \textit{WRITE TITLE} \textit{ syntax})$
--

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	N P I	ja	nein

Verwandte Statements

AT TOP OF PAGE, AT END OF PAGE.

Funktion

Das Statement NEWPAGE dient dazu, einen Seitenvorschub auszulösen und eine neue Seite zu beginnen. NEWPAGE bewirkt außerdem, daß etwaige AT END OF PAGE- und WRITE TRAILER-Statements ausgeführt werden. Ist eine Seitenüberschrift-Verarbeitung aber nicht ausdrücklich definiert (WRITE TITLE, WRITE NOTITLE oder DISPLAY NOTITLE), erhält jede neue Seite einen Standardüberschrift mit Datum, Uhrzeit und laufender Seitennummer.

Anmerkungen:

- ① *Der Seitenvorschub wird dann nicht vorgenommen, wenn das NEWPAGE-Statement ausgeführt wird, sondern nur, wenn ein anschließendes, eine Ausgabe erzeugendes Statement ausgeführt wird.*
- ② *Wird kein NEWPAGE-Statement verwendet, so wird der Seitenvorschub automatisch in Abhängigkeit von der mit dem Session-Parameter PS definierten Seitenlänge gesteuert.*

Report-Spezifikation (*rep*)

Mit der Notation (*rep*) kann ein bestimmter anderer Report angegeben werden, auf den sich das Statement beziehen soll. Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden.

Falls nichts anderes angegeben wird, bezieht sich das NEWPAGE-Statement auf den ersten Report (Report 0).

EVEN IF TOP OF PAGE

Mit dieser Option bewirken Sie, daß das NEWPAGE-Statement einen Seitenvorschub (mit entsprechender AT TOP OF PAGE- und Seitenüberschrift-Verarbeitung) auslöst, auch wenn unmittelbar vorher bereits ein Seitenvorschub erfolgt ist.

WHEN LESS THAN *operand1* LINES LEFT

Mit dieser Option bewirken Sie, daß das NEWPAGE-Statement ausgeführt wird, wenn auf der aktuellen Seite weniger als *operand1* Zeilen übrig sind. Der interne Zeilenzähler orientiert sich hierbei am Wert des Session-Parameters PS.

WITH TITLE

Die Option WITH TITLE können Sie verwenden, um auf der generierten neuen Seite eine Überschrift auszugeben. Für die WITH TITLE-Option gilt dieselbe Syntax wie für das Statement WRITE TITLE, außer daß die SKIP-Klausel in einem NEWPAGE WITH TITLE-Statement nicht erlaubt ist.

Beispiel

```

/* EXAMPLE 'NWPEX1S': NEWPAGE
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 SALARY (1)
  2 CURR-CODE (1)
END-DEFINE
/*****
LIMIT 15
READ EMPLOY-VIEW BY CITY FROM 'DENVER'
  DISPLAY CITY (IS=ON) NAME SALARY (1) CURR-CODE (1)
  AT BREAK OF CITY
  SKIP 1
/*****
NEWPAGE WHEN LESS THAN 10 LINES LEFT
WRITE '*****'
  / 'SUMMARY FOR ' OLD(CITY)
  / '*****'
  / '*****'
  / 'SUM OF SALARIES:' SUM(SALARY(1))
  / 'AVG OF SALARIES:' AVER(SALARY(1))
  / '*****'
NEWPAGE
/*****
END-BREAK
END-READ
END

```

PAGE	1	97-03-27 15:21:13		
CITY	NAME	ANNUAL SALARY	CURRENCY CODE	
DENVER	TANIMOTO	33000	USD	
	MEYER	50000	USD	

SUMMARY FOR DENVER				

SUM OF SALARIES:		83000		
AVG OF SALARIES:		41500		

PAGE		2		97-03-27 15:21:13	
CITY	NAME	ANNUAL SALARY	CURRENCY CODE		
DERBY	DEAKIN	7950	UKL		
	GARFIELD	7200	UKL		
	MUNN	8200	UKL		
	MUNN	5200	UKL		
	GREBBY	8450	UKL		
	WHITT	7450	UKL		
	PONSONBY	4450	UKL		
	MAGUIRE	4000	UKL		
	HEYWOOD	3800	UKL		
	BRYDEN	6450	UKL		
	SMITH	29000	UKL		
	CONQUEST	34000	UKL		
	ACHIESON	10500	UKL		

PAGE		3		97-03-27 15:21:13	
CITY	NAME	ANNUAL SALARY	CURRENCY CODE		

SUMMARY FOR DERBY					

SUM OF SALARIES:		136650			
AVG OF SALARIES:		10511			

Äquivalentes Reporting-Mode-Beispiel: siehe Programm NWPEX1R in Library SYSEXRM.

NOTITLE...

Im folgenden finden Sie eine Liste der auf NOTITLE... bezogenen Statements:

- DISPLAY
- PRINT
- WRITE

OBTAIN

Anmerkung:

Dieses Statement kann nur im Reporting Mode verwendet werden.

OBTAIN *operand1...*

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S A G	A N P I F B D T L	ja	ja

Funktion

Das Statement OBTAIN dient dazu, Datenbankfelder zu lesen oder einen Bereich von Ausprägungen eines Datenbank-Arrays, der in einem zusammenhängenden Speicherbereich gespeichert ist. Es können auch mehrere Bereiche von Ausprägungen gelesen werden.

Werden mehrere Bereiche angesprochen und soll in einem späteren Statement einer dieser Bereiche referenziert werden, so dient bei der Referenzierung der Anfang des jeweiligen Index zur Identifizierung eines bestimmten Bereichs. Die Ausprägung, mit der ein bestimmter Bereich beginnt, wird hierbei immer als Ausprägung 1 gezählt.

operand1

Als *operand1* geben Sie die Felder an, die mit dem OBTAIN-Statement gelesen werden sollen.

Beispiele

Siehe Programme OBTEX1 und OBTEX2 in Library SYSEXRM.

ON ERROR

Structured-Mode-Syntax

```
ON ERROR  
  statement...  
END-ERROR
```

Reporting-Mode-Syntax

```
ON ERROR { statement  
           DO statement... DOEND }
```

Funktion

Das Statement ON ERROR dient dazu, Laufzeitfehler abzufangen, welche andernfalls eine Natural-Fehlermeldung, den Abbruch des gerade ausgeführten Programms und die Rückkehr zum Kommandoeingabe-Modus zur Folge hätten.

Sobald die Ausführung der in einem ON ERROR-Statement-Block angegebenen Statements beginnt, ist der normale Programmablauf unterbrochen und kann nicht fortgesetzt werden. Eine Ausnahme bildet Fehler 3145 (angeforderter Datensatz im “Hold”), bei dem über ein RETRY-Statement die Verarbeitung genau an der Stelle, an der sie unterbrochen wurde, fortgesetzt werden kann.

Dieses Statement ist non-prozedural (das heißt, seine Ausführung hängt von einem Ereignis ab, nicht davon, wo im Programm es steht).

ON ERROR-Verarbeitung in Unterprogrammen

Wird mittels CALLNAT, PERFORM oder FETCH RETURN eine Unterprogramm-Struktur aufgebaut, so darf jedes Modul dieser Struktur ein ON ERROR-Statement enthalten.

Tritt ein Fehler auf, so verfolgt Natural die Unterprogramm-Struktur automatisch zurück und wählt das erste in einem Unterprogramm gefundene ON ERROR-Statement zur Ausführung aus. Enthält keines der Module ein ON ERROR-Statement, gibt Natural eine entsprechende Fehlermeldung aus, und es erfolgt — wie oben beschrieben — ein Programmabbruch.

Einschränkung

Jedes Natural-Objekt darf maximal ein ON ERROR-Statement enthalten.

Systemvariablen *ERROR-NR und *ERROR-LINE

Die Natural-Systemvariable *ERROR-NR enthält die Fehlernummer des von Natural entdeckten Fehlers.

Die Natural-Systemvariable *ERROR-LINE enthält die Nummer der Sourcecode-Zeile, in der das Statement steht, das den Fehler verursacht hat.

Verlassen eines ON ERROR-Blocks

Ein ON ERROR-Statement-Block kann mit einem der Statements FETCH, STOP, TERMINATE, RETRY oder ESCAPE ROUTINE verlassen werden. Wird der Block nicht mit einem dieser Statements verlassen, gibt Natural eine entsprechende Fehlermeldung aus und bricht die Ausführung des Programms ab.

Beispiel

```

/* EXAMPLE 'ONEEX1': ON ERROR
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
1 #NAME (A20)
1 #CITY (A20)
END-DEFINE
/*****
REPEAT
  INPUT 'ENTER NAME:' #NAME
  IF #NAME = ' '
    STOP
  END-IF
  FIND EMPLOY-VIEW WITH NAME = #NAME
  INPUT (AD=M) 'ENTER NEW VALUES:' ///
    'NAME:' NAME /
    'CITY:' CITY

  UPDATE
  END TRANSACTION
/*****
ON ERROR
  IF *ERROR-NR = 3009
    WRITE 'LAST TRANSACTION NOT SUCCESSFUL'
    / 'HIT ENTER TO RESTART PROGRAM'
    FETCH 'PROGUPD'
  END-IF
END-ERROR
/*****
  END-FIND
END-REPEAT
END

```

OPEN CONVERSATION

OPEN CONVERSATION USING [SUBPROGRAMS] {operand1} ...

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S A	A	ja	nein

Verwandte Statements

CLOSE CONVERSATION, DEFINE DATA CONTEXT.

Funktion

Das Statement OPEN CONVERSATION wird im Zusammenhang mit Natural Remote Procedure Call (RPC) verwendet. Es ermöglicht dem Client, eine Conversation zu öffnen und die Remote-Subprogramme anzugeben, die an der Conversation beteiligt sein sollen.

Wenn das OPEN CONVERSATION-Statement ausgeführt wird, weist es der Systemvariablen *CONVID eine eindeutige ID zu, die die Conversation identifiziert.

Subprogramm-Namen (*operand1*)

Als *operand1* geben Sie die Namen der Subprogramme an, die an der Conversation beteiligt sein sollen. Der Name eines Subprogramms kann entweder als 1 bis 8 Zeichen lange Konstante oder als alphanumerische Variable mit Länge 1 bis 8 angegeben werden.

Weitere Informationen und Beispiele

Siehe Beschreibung von Natural RPC in Ihrer *Natural RPC Documentation* bzw. in Ihrer *Natural Installation and Operations Documentation*.

OPEN DIALOG

$$\text{OPEN DIALOG } \textit{operand1} \text{ [USING] [PARENT] } \textit{operand2} \\
 \text{ [[GIVING] [DIALOG-ID] } \textit{operand3} \text{]} \\
 \left[\text{ WITH } \left\{ \left(\textit{operand4} \left[\text{ (AD= } \left\{ \begin{matrix} \text{M} \\ \text{O} \\ \text{A} \end{matrix} \right\} \right] \right) \right\} \dots \right\} \right] \\
 \text{ nX } \\
 \text{ PARAMETERS-clause }$$

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	ja	nein
Operand2	C S	* G	nein	nein
Operand3	S	I	ja	nein
Operand4	C S A	A N P I F B D T L C G O	ja	nein

* Handle

Anmerkung:

Dieses Statement ist nur unter Windows und Windows NT verfügbar.

Verwandtes Statement

CLOSE DIALOG.

Funktion

Dieses Statement dient dazu, einen Dialog dynamisch zu öffnen.

Dialog-Name (*operand1*)

Operand1 ist der Name des Dialogs, der geöffnet wird.

Wenn die *PARAMETERS-Klausel* verwendet wird, muß *operand1* eine Konstante sein.

Handle-Name (*operand2*)

Operand2 ist der "Handle"-Name des "Parents".

Dialog-ID (*operand3*)

Operand3 ist eine eindeutige Kennung, die bei der Erstellung des Dialogs zurückgegeben wird.

Operand3 muß mit Format/Länge I4 definiert werden.

AD=

Wenn *operand4* eine Variable ist, können Sie sie auf eine drei folgenden Arten kennzeichnen:

Variable	Beschreibung
AD=O	nicht änderbar
AD=M	änderbar
AD=A	nur Eingabe

Die Voreinstellung für AD= ist AD=M.

Operand4 kann nicht explizit angegeben werden, wenn *operand4* eine Konstante ist. AD=O gilt immer für Konstanten.

AD=M

Der übergebene Wert eines Parameters kann standardmäßig im Dialog geändert werden, und der geänderte Wert kann zurück zum aufrufenden Objekt übergeben werden, wo er den ursprünglichen Wert überschreibt.

Ausnahme: Für ein in der Parameter Data Area des Dialogs mit BY VALUE definiertes Feld wird kein Wert zurückgegeben.

AD=O

Wenn Sie einen Parameter mit AD=O kennzeichnen, kann der übergebene Wert im Dialog geändert werden, der geänderte Wert kann aber nicht zum aufrufenden Objekt zurückgegeben werden; d.h. das Feld im aufrufenden Objekt behält seinen ursprünglichen Wert bei.

Anmerkung:

Intern wird AD=O auf dieselbe Art und Weise wie BY VALUE verarbeitet (siehe den Abschnitt Parameter-Data-Definition in der Beschreibung des DEFINE DATA-Statements).

AD=A

Wenn Sie einen Parameter mit AD=A kennzeichnen, wird sein Wert nicht an den Dialog übergeben, sondern er erhält einen Wert vom Dialog. AD=A-Felder werden auf Leerzeichen zurückgesetzt, bevor der Dialog geöffnet wird.

Für ein in der Parameter Data Area des Dialogs mit BY VALUE definiertes Feld kann das aufrufende Objekt keinen Wert erhalten. In diesem Fall bewirkt AD=A lediglich, daß das Feld auf Leerzeichen zurückgesetzt wird, bevor der Dialog aufgerufen wird.

Übergabe von Parametern an den Dialog

Beim Öffnen eines Dialogs können Parameter an den Dialog übergeben werden.

Als *operand4* geben Sie die Parameter an, die an den Dialog übergeben werden sollen.

Mit der *PARAMETERS-clause* können Parameter selektiv übergeben werden.

nX

Mit der Notation *nX* können Sie angeben, daß die nächsten *n* Parameter übersprungen werden sollen (z.B., 1X, um den nächsten Parameter zu überspringen, oder 3X, um die nächsten drei Parameter zu überspringen); dies bedeutet, daß für die nächsten *n* Parameter keine Werte an den Dialog übergeben werden.

Ein Parameter, der übersprungen werden soll, muß mit dem Schlüsselwort OPTIONAL im DEFINE DATA PARAMETER-Statement des Dialogs definiert werden. OPTIONAL bedeutet, daß ein Wert vom aufrufenden Objekt an einen solchen Parameter übergeben werden kann — aber nicht unbedingt muß.

PARAMETERS-clause

```
PARAMETERS {parameter-name = operand4} ... END-PARAMETERS
```

Anmerkung:

Sie können die PARAMETERS-clause nur verwenden, wenn operand1 eine Konstante ist und der Dialog katalogisiert ist.

Parameter-name ist der Name des Parameters, wie im Parameter-Data-Area-Teil des Dialogs definiert.

Anmerkung:

Wenn der Wert eines mit AD=O gekennzeichneten, "by reference" übergebenen Parameters in einem Dialog geändert wird, führt dies zu einem Laufzeitfehler.

Weitere Informationen und Beispiele

Siehe Kapitel **Event-Driven Programming Techniques** im *Natural User's Guide for Windows*.

OPTIONS

Dieses Statement ist nur auf Großrechnern verfügbar.

Das Statement OPTIONS kann verwendet werden, um verschiedene Kompilierungs-Optionen für das aktuelle Natural-Programmierobjekt anzugeben. Dieselben Optionen können verwendet werden, um Angaben wie folgt vorzunehmen:

- statisch mit dem Makro NTCMPO
- dynamisch mit dem Parameter CMPO
- innerhalb einer Natural-Session mit dem Systemkommando COMPOPT.

Außerdem kann das Statement OPTIONS verwendet werden, um Optionen für den Natural Optimizer Compiler anzugeben. Diese Optionen sind in der *Natural Optimizer Compiler Documentation* beschrieben.

Mit der Option MCG angegebene Natural Optimizer Compiler-Optionen werden auf Gültigkeit hin überprüft, auch wenn der Natural Optimizer Compiler nicht installiert ist.

Wenn mehrere OPTIONS-Statements innerhalb desselben Programmierobjekts angegeben sind, erlangen die Optionseinstellungen sofort Gültigkeit. Dies ist bei den Optionen PSIGNE, TSENABL und GFID allerdings nicht der Fall. Bei diesen Optionen gilt der mit dem **letzten** OPTIONS-Statement angegebene Optionswert.

PASSW

PASSW=*operand1*

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	ja	nein

Funktion

Mit dem Statement **PASSW** geben Sie ein Paßwort an, um auf eine passwortgeschützte Adabas- oder VSAM-Datei zugreifen zu können.

Paßwort – operand1

Das Paßwort (*operand1*) kann entweder als alphanumerische Konstante angegeben werden oder als alphanumerische Variable, welche das Paßwort enthält. Das Paßwort darf bis zu acht Zeichen lang sein und keine Sonderzeichen oder Leerzeichen enthalten. Wird es als Konstante angegeben, muß es in Apostrophen stehen.

Das mit dem **PASSW**-Statement angegebene Paßwort gilt für alle Datenbankzugriffs-Statements (FIND, GET, HISTOGRAM, READ, STORE), in denen kein eigenes Paßwort angegeben ist. Es gilt, bis mit einem anderen **PASSW**-Statement ein anderes Paßwort angegeben wird bzw. bis zum Ende der Natural-Session.

Ein mit einem bestimmten Datenbank-Statement angegebenes Paßwort gilt nur für das jeweilige Statement, nicht für irgendwelche nachfolgenden Statements.

Hinweise zu Natural Security

Im Security-Profil einer Library können Sie ein standardmäßiges Adabas-Paßwort angeben (wie in der *Natural Security Documentation* beschrieben); dieses Paßwort gilt für alle Datenbankzugriffs-Statements, für die weder ein eigenes Paßwort angegeben ist noch ein **PASSW**-Statement gilt, und zwar nicht nur in der betreffenden Library, sondern darüber hinaus auch, wenn Sie anschließend in andere Libraries wechseln, in deren Security-Profilen kein Paßwort angegeben ist.

Einschränkung

Dieses Statement gilt nicht für DL/I-Datenbanken.

Unterdrückung der Paßwort-Anzeige (nur auf Großrechnern)

Wird das Paßwort als Konstante angegeben, sollte das Statement ganz am Anfang einer Sourcecode-Zeile stehen und das Gleichheitszeichen “=” unmittelbar auf PASSW folgen. Dadurch ist gewährleistet, das das Paßwort im Sourcecode nicht sichtbar ist.

Im Online-Betrieb können Sie das PASSW-Statement unsichtbar eingeben, indem Sie zuerst das Terminalkommando “%*” eingeben, bevor Sie das PASSW-Statement eintippen.

Im Batch-Betrieb kann ein Paßwort folgendermaßen angegeben werden:

```
ADHOC  
PASSW='password'  
END  
ENDHOC
```

Das Paßwort wird dann bei der Ausgabe nicht ausgedruckt.

Beispiel

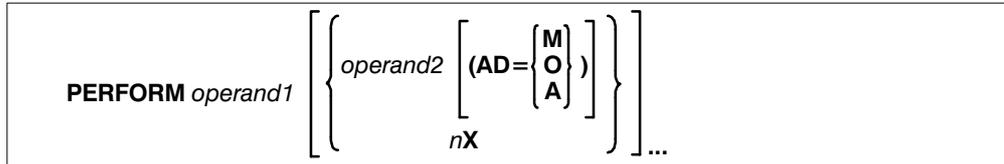
```

/* EXAMPLE 'PWDEX1:' PASSW
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
END-DEFINE
/*****
PASSW= 'PASSW1'
/*****
LIMIT 5
READ EMPLOY-VIEW
  DISPLAY NOTITLE PERSONNEL-ID NAME
END-READ
/*****
END

```

PERSONNEL ID	NAME
50005600	MORENO
50005500	BLOND
50005300	MAIZIERE
50004900	CAUDAL
50004600	VERDIE

PERFORM



Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C	A	nein	nein
Operand2	C S A G	A N P I F B D T L C G O	ja	ja

Verwandte Statements

DEFINE SUBROUTINE, DEFINE DATA PARAMETER, CALLNAT, FETCH.

Funktion

Das Statement PERFORM dient dazu, eine Natural-Subroutine aufzurufen.

operand1

Als *operand1* geben Sie den Namen der aufzurufenden Subroutine an. Die Subroutine muß in einem DEFINE SUBROUTINE-Statement definiert sein. Sie kann entweder als interne oder als externe Subroutine definiert werden (vgl. DEFINE SUBROUTINE-Statement).

Innerhalb eines Objekts können maximal 50 externe Subroutinen referenziert werden.

Von einer Subroutine benutzbare Daten

Interne Subroutinen

Es ist nicht möglich, mit dem PERFORM-Statement irgendwelche Parameter explizit vom aufrufenden Programm an eine programmintern definierte Subroutine zu übergeben.

Eine programminterne Subroutine kann auf die im selben Objektmodul verwendete Local Data Area sowie auf die derzeit verwendete Global Data Area zugreifen.

Externe Subroutinen

Eine programmextern definierte Subroutine kann Daten aus der Global Data Area des aufrufenden Objekts verwenden. Außerdem können Sie mit dem PERFORM-Statement Parameter vom aufrufenden Objekt an die aufgerufene Subroutine übergeben (siehe *operand2*); dadurch können Sie die Größe der Global Data Area entsprechend klein halten.

Parameter (*operand2*)

Wenn Sie mit dem PERFORM-Statement eine *externe* Subroutine aufrufen, können Sie mit dem PERFORM-Statement einen oder mehrere Parameter (*operand2*) vom aufrufenden Objekt an die externe Subroutine übergeben. Für *interne* Subroutinen kann *operand2* nicht angegeben werden.

Wenn Parameter übergeben werden, muß die Struktur der Parameterliste in einem DEFINE DATA-Statement definiert werden.

Standardmäßig erfolgt die Übergabe der Parameter “by reference”, d.h. die Daten werden über Adreß-Parameter übergeben, die Parameterwerte selbst werden nicht übertragen.

Es besteht aber auch die Möglichkeit, Parameter “by value” zu übergeben, d.h. die Parameterwerte selbst zu übergeben. Hierzu definieren Sie die betreffenden Felder im DEFINE DATA PARAMETER-Statement der Subroutine mit der Option BY VALUE bzw. BY VALUE RESULT).

- Für die Parameterübergabe “by reference” gilt: Reihenfolge, Format und Länge der Parameter im aufrufenden Objekt müssen genau den Angaben im DEFINE DATA PARAMETER-Statement der aufgerufenen Subroutine entsprechen. Die Namen der Variablen im aufrufenden Objekt und der Subroutine können unterschiedlich sein.
- Für die Parameterübergabe “by value” gilt: Die Reihenfolge der Parameter im aufrufenden Objekt muß der Reihenfolge im DEFINE DATA PARAMETER-Statement der aufgerufenen Subroutine entsprechen. Formate und Längen der Variablen im aufrufenden Objekt und in der Subroutine können unterschiedlich sein, müssen aber datenübertragungskompatibel sein. Die Namen der Variablen im aufrufenden Objekt und in der Subroutine können unterschiedlich sein.

Um Parameterwerte, die in der Subroutine verändert wurden, an das aufrufende Objekt zurückgeben zu können, müssen Sie die betreffenden Felder mit BY VALUE RESULT definieren.

Mit BY VALUE (ohne RESULT) ist es nicht möglich, veränderte Parameterwerte an das aufrufende Objekt zurückzugeben (unabhängig von der “AD”-Angabe; vgl. unten).

Anmerkung:

Intern wird bei BY VALUE eine Kopie der Parameterwerte erzeugt. Die Subroutine greift auf diese Kopie zu und kann sie modifizieren, was aber keinen Einfluß auf die Originalparameterwerte im aufrufenden Objekt hat.

Bei BY VALUE RESULT wird ebenfalls eine Kopie erzeugt, aber nach Beendigung der Subroutine überschreiben die (modifizierten) Werte der Kopie die Originalparameterwerte.

Für beide Arten der Parameterübergabe sind folgende Punkte zu beachten:

Eine Gruppe darf in der Parameter Data Area der aufgerufenen Subroutine *innerhalb* eines REDEFINE-Blocks redefiniert werden.

Bei der Übergabe eines Arrays muß die Anzahl seiner Dimensionen und Ausprägungen in der Parameter Data Area der Subroutine denen in der PERFORM-Parameterliste entsprechen.

Anmerkung:

Wenn mehrere Ausprägungen eines Arrays, das als Teil einer indizierten Gruppe definiert ist, mit dem PERFORM-Statement übergeben werden, dürfen die entsprechenden Felder in der Parameter Data Area der Subroutine nicht redefiniert werden, da sonst die falschen Adressen übergeben werden.

AD=

Wenn *operand2* eine Variable ist, können Sie sie folgendermaßen kennzeichnen:

AD=O	nicht modifizierbar
AD=M	modifizierbar
AD=A	nur für Eingabe

Standardmäßig gilt AD=M.

Wenn *operand2* eine Konstante ist, kann AD nicht explizit angegeben werden. Für Konstanten gilt immer AD=O.

AD=M

Standardmäßig kann der übergebene Wert eines Parameters in der Subroutine geändert und der geänderte Wert an das aufrufende Objekt zurückgegeben werden, wo er den ursprünglichen Wert überschreibt.

Ausnahme: Bei einem Feld, das in der Parameter Data Area der Subroutine mit BY VALUE definiert ist, wird kein Wert zurückgegeben.

AD=O

Wenn Sie einen Parameter mit AD=O kennzeichnen, kann der übergebene Wert in der Subroutine geändert, aber nicht an das aufrufende Objekt zurückgegeben werden; d.h. im aufrufenden Objekt behält das Feld seinen ursprünglichen Wert.

Anmerkung:

Intern wird AD=O wie BY VALUE verarbeitet (vgl. Anmerkung unter Parameter – operand 2 auf Seite 459).

AD=A

Wenn Sie einen Parameter mit AD=A kennzeichnen, wird sein Wert nicht *an* die Subroutine übergeben, sondern er wird auf Leerwert zurückgesetzt, bevor die Subroutine aufgerufen wird, und kann dazu verwendet werden, einen Wert *von* der Subroutine zu erhalten.

Bei einem Feld, das in der Parameter Data Area der Subroutine mit BY VALUE definiert ist, kann das aufrufende Objekt keinen Wert erhalten. Hier bewirkt AD=A lediglich, daß das Feld vor dem Aufruf auf Leerwert zurückgesetzt wird.

nX

Diese Notation steht auf Großrechnern nicht zur Verfügung.

Mit der Notation *nX* können Sie angeben, daß die nächsten *n* Parameter übersprungen werden sollen (z.B. 1X, um den nächsten Parameter zu überspringen, oder 3X, um die nächsten drei Parameter zu überspringen), dies bedeutet, daß für die nächsten *n* Parameter keine Werte an die externe Subroutine übergeben werden.

Ein zu überspringender Parameter muß mit dem Schlüsselwort **OPTIONAL** im Statement **DEFINE DATA PARAMETER** der Subroutine definiert werden. **OPTIONAL** bedeutet, daß ein Wert vom aufrufenden Objekt an einen solchen Parameter übergeben werden kann — aber nicht unbedingt muß.

Verschachtelte **PERFORM**-Statements

Eine aufgerufene Subroutine kann ihrerseits mit einem **PERFORM**-Statement eine andere Subroutine aufrufen. Wieviele Ebenen eine derartige Verschachtelung mehrerer **PERFORM**-Statements erreichen darf, hängt vom benötigten Speicherplatz ab.

Eine Subroutine kann auch sich selbst aufrufen (rekursive Subroutine). Falls eine rekursive externe Subroutine Datenbankzugriffe beinhaltet, sorgt Natural automatisch dafür, daß diese als getrennte logische Operationen durchgeführt werden.

Parameter-Übertragung mit dynamischen Variablen

Siehe **CALLNAT**.

Beispiel 1

```

/* EXAMPLE 'PEREX1S': PERFORM (STRUCTURED MODE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE (A20/2)
  2 PHONE
1 #ARRAY (A75/1:4)
1 REDEFINE #ARRAY
  2 #ALINE (A25/1:4,1:3)
1 #X (N2) INIT <1>
1 #Y (N2) INIT <1>
END-DEFINE
/*****
LIMIT 5
FIND EMPLOY-VIEW WITH CITY = 'BALTIMORE'
  MOVE NAME TO #ALINE (#X,#Y)
  MOVE ADDRESS-LINE(1) TO #ALINE (#X+1,#Y)
  MOVE ADDRESS-LINE(2) TO #ALINE (#X+2,#Y)
  MOVE PHONE TO #ALINE (#X+3,#Y)
  IF #Y = 3
    RESET INITIAL #Y
    PERFORM PRINT
  ELSE
    ADD 1 TO #Y
  END-IF
AT END OF DATA
  PERFORM PRINT
END-ENDDATA
END-FIND
/*****
DEFINE SUBROUTINE PRINT
  WRITE NOTITLE (AD=OI) #ARRAY(*)
  RESET #ARRAY(*)
  SKIP 1
END-SUBROUTINE
/*****
END

```

JENSON 2120 HASSELL #206 (301)998-5038	LAWLER 4588 CANDLEBERRY AVE BALTIMORE (301)629-0403	FORREST 37 TENNYSON DRIVE BALTIMORE (301)881-3609
ALEXANDER 409 SENECA DRIVE BALTIMORE (301)345-3690	NEEDHAM 12609 BUILDERS LANE BALTIMORE (301)641-9789	

Beispiel 2

Programm mit PERFORM-Statement:

```

/* EXAMPLE 'PEREX2' PERFORM EXTERNAL WITH PARAMETER
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE (A20/2)
  2 PHONE
1 #ALINE (A25/1:4,1:3)
1 #X (N2) INIT <1>
1 #Y (N2) INIT <1>
END-DEFINE
/*****
LIMIT 5
FIND EMPLOY-VIEW WITH CITY = 'BALTIMORE'
MOVE NAME TO #ALINE (#X,#Y)
MOVE ADDRESS-LINE(1) TO #ALINE (#X+1,#Y)
MOVE ADDRESS-LINE(2) TO #ALINE (#X+2,#Y)
MOVE PHONE TO #ALINE (#X+3,#Y)
IF #Y = 3
DO
  RESET INITIAL #Y
  PERFORM PEREX2E #ALINE(*,*)
DOEND
ELSE
  ADD 1 TO #Y
AT END OF DATA
  PERFORM PEREX2E #ALINE(*,*)
LOOP
/*****
END

```

Aufgerufene Subroutine:

```

/* EXAMPLE 'PEREX2E' SUBROUTINE WITH PARAMETER
/*****
DEFINE DATA PARAMETER
1 #ALINE (A25/1:4,1:3)
END-DEFINE
/*****
DEFINE SUBROUTINE PEREX2E
  WRITE NOTITLE (AD=OI) #ALINE(*,*)
  RESET #ALINE(*,*)
  SKIP 1
RETURN
/*****
END

```

JENSON 2120 HASSELL #206 (301)998-5038	LAWLER 4588 CANDLEBERRY AVE BALTIMORE (301)629-0403	FORREST 37 TENNYSON DRIVE BALTIMORE (301)881-3609
ALEXANDER 409 SENECA DRIVE BALTIMORE (301)345-3690	NEEDHAM 12609 BUILDERS LANE BALTIMORE (301)641-9789	

PERFORM BREAK PROCESSING

PERFORM BREAK [PROCESSING] [(r)]
AT BREAK *statement...*

Verwandtes Statement

AT BREAK.

Funktion

Das Statement PERFORM BREAK PROCESSING dient dazu, bei Verarbeitungsschleifen, die mit FOR, REPEAT, CALL LOOP oder CALL FILE ausgelöst wurden, eine Gruppenwechsel-Verarbeitung auszulösen, wo keine automatische Gruppenwechsel-Verarbeitung durchgeführt wird, oder wenn eine Gruppenwechsel-Verarbeitung gewünscht wird. Im Gegensatz zu einer automatischen Gruppenwechsel-Verarbeitung, die ausgeführt wird, unmittelbar nachdem der Datensatz gelesen wurde, wird ein PERFORM BREAK-Statement dann ausgeführt, wenn es im normalen Programmablauf auftaucht.

Das PERFORM BREAK-Statement überprüft anhand des Wertes eines Kontrollfeldes, ob eine Gruppenwechsel-Bedingung erfüllt wird, und bewirkt außerdem eine Auswertung der Natural-Systemfunktionen. Diese Prüfung und Auswertung findet jedesmal, wenn das Statement ausgeführt wird, statt. Die Ausführung eines PERFORM BREAK-Statements kann an eine mit einem IF-Statement angegebene logische Bedingung geknüpft werden.

Statement-Referenzierung (r)

Normalerweise wird die PERFORM BREAK-Verarbeitung zum letztenmal ausgeführt, wenn die Ausführung des Programms/Subprogramms bzw. der Subroutine beendet ist.

Durch Verwendung eines Statement-Labels oder Angabe der Sourcecode-Zeilenummer (r) kann eine bestimmte Verarbeitungsschleife referenziert werden, auf die sich die abschließende PERFORM BREAK-Verarbeitung beziehen soll; in diesem Falle ist sie Teil der schleifenbeendenden Verarbeitung, d.h. die letzte PERFORM BREAK-Verarbeitung wird nach der letzten automatischen Gruppenwechsel-Verarbeitung und vor den AT END OF DATA-Statements ausgeführt.

AT BREAK *statement...*

Siehe Syntax des AT BREAK-Statements.

Beispiel

```

/* EXAMPLE 'PBPEX1': PERFORM BREAK PROCESSING
/*****
DEFINE DATA LOCAL
1 #INDEX (N2)
1 #LINE (N2) INIT <1>
END-DEFINE
/*****
FOR #INDEX 1 TO 18
PERFORM BREAK PROCESSING
  AT BREAK OF #INDEX /1/
  WRITE NOTITLE / 'PLEASE COMPLETE LINES 1-9 ABOVE' /
  RESET INITIAL #LINE
  END-BREAK
  WRITE NOTITLE ' _' (64) '=' #LINE
  ADD 1 TO #LINE
END-FOR
/*****
END

```

	#LINE:	1
	#LINE:	2
	#LINE:	3
	#LINE:	4
	#LINE:	5
	#LINE:	6
	#LINE:	7
	#LINE:	8
	#LINE:	9
PLEASE COMPLETE LINES 1-9 ABOVE		
	#LINE:	1
	#LINE:	2
	#LINE:	3
	#LINE:	4
	#LINE:	5
	#LINE:	6
	#LINE:	7
	#LINE:	8
	#LINE:	9
PLEASE COMPLETE LINES 1-9 ABOVE		

Äquivalentes Reporting-Mode-Beispiel: siehe Programm PBPEX1R in Library SYSEXRM.

PRINT

PRINT [(*rep*)] [**NOTITLE**] [**NOHDR**] [(*statement-parameters*)]

$$\left\{ \begin{array}{l} nX \\ nT \\ / \end{array} \right\} \dots \left\{ \begin{array}{l} \text{'text' [(attributes)]} \\ \text{'c'(n) [(attributes)]} \\ \text{operand1 [(parameters)]} \end{array} \right\} \dots$$

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S A G N	A N P I F B D T L G O	ja	nein

Verwandtes Statement

WRITE.

Funktion

Das Statement PRINT dient dazu, Ausgaben im freien Format zu erzeugen.

Das PRINT-Statement unterscheidet sich vom WRITE-Statement in folgenden Punkten:

- Die Ausgabelänge der einzelnen Operanden ergibt sich aus der Länge der tatsächlich ausgegebenen Werte und nicht aus der Länge der verwendeten Felder. Vorangestellte Nullen (bei numerischen Werten) und nachgestellte Leerzeichen (bei alphanumerischen Werten) werden nicht mit ausgegeben.
Mit dem Session-Parameter AD können Sie festlegen, ob numerische Werte links- oder rechtsbündig ausgegeben werden sollen: mit AD=L werden einem numerischen Wert nachfolgende Leerstellen nicht ausgegeben; mit AD=R werden einem numerischen Wert vorangestellte Leerzeichen mit ausgegeben.
- Überschreitet die Ausgabe die vorgegebene Zeilenlänge (Parameter LS), wird die Ausgabe in der nächsten Zeile wie folgt fortgesetzt:
Eine alphanumerische Konstante oder der Inhalt einer alphanumerischen Variablen (ohne Editiermaske) wird ab dem letzten auf der aktuellen Zeile ausgegebenen Leerzeichen oder Zeichen, das weder ein Buchstabe noch eine Ziffer ist, abgetrennt. Der erste Teil des Wertes verbleibt auf der aktuellen Zeile, der abgetrennte Teil wird in der nächsten Zeile ausgegeben.
Bei allen anderen Operanden wird der gesamte Wert, der nicht mehr in die aktuelle Zeile paßt, in der nächsten Zeile ausgegeben.

Report-Spezifikation (*rep*)

Mit der Notation (*rep*) kann ein bestimmter anderer Report angegeben werden, auf den sich das Statement beziehen soll. Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden. Falls nichts anderes angegeben wird, bezieht sich das PRINT-Statement auf den ersten Report (Report 0).

NOTITLE

Für jede über ein PRINT-Statement ausgegebene Seite generiert Natural eine Titelzeile, die die laufende Seitennummer, die Uhrzeit und das Datum enthält. Die Uhrzeit wird zu Beginn der Session (TP-Betrieb) oder zu Beginn des Jobs (Batch-Betrieb) gesetzt. Die generierte Titelzeile kann entweder durch eine eigene mit einem WRITE TITLE-Statement angegebene Titelzeile überschrieben oder durch eine NOTITLE-Klausel im PRINT-Statement unterdrückt werden.

PRINT NAME	(generierte Titelzeile wird ausgegeben)
PRINT NAME	
WRITE TITLE 'USER TITLE'	(eigene Titelzeile wird ausgegeben)
PRINT NOTITLE NAME	(keine Titelzeile wird ausgegeben)

Wenn die NOTITLE-Option verwendet wird, gilt sie für alle DISPLAY-, PRINT- und WRITE-Statements im selben Objekt, die Daten auf denselben Report schreiben.

NOHDR

Das PRINT-Statement selbst erzeugt keine Spaltenüberschriften. Wenn Sie allerdings das PRINT-Statement zusammen mit einem DISPLAY-Statement verwenden, können Sie mit der Option NOHDR des PRINT-Statements die vom DISPLAY-Statement generierten Spaltenüberschriften unterdrücken: Die NOHDR-Option ist nur relevant, wenn das PRINT-Statement *nach* einem DISPLAY-Statement steht, die Ausgabe sich insgesamt über mehr als eine Seite erstreckt und die Ausführung des PRINT-Statements zur Ausgabe einer neuen Seite führt. Ohne NOHDR-Option würden auf dieser neuen Seite die DISPLAY-Spaltenüberschriften ausgegeben, mit NOHDR werden sie dort nicht ausgegeben.

statement-parameters

Unmittelbar nach dem PRINT-Schlüsselwort selbst oder nach einem der auszugebenden Felder können in Klammern Session-Parameter gesetzt werden.

Diese Parameter haben dann für das jeweilige Statement oder Feld Gültigkeit statt der betreffenden mit einem GLOBALS-Kommando, SET GLOBALS- oder FORMAT-Statement gesetzten Parameter. Werden mehrere Parameter angegeben, müssen sie jeweils durch ein oder mehrere Leerzeichen voneinander getrennt werden. Die Angabe eines Parameters darf sich nicht über zwei Sourcecode-Zeilen erstrecken.

Die hier gültigen Parameter-Einstellungen kommen nur für Variablen-Felder in Betracht, haben aber keine Auswirkungen auf Text-Konstanten. Wenn Sie Feldattribute für eine Text-Konstante setzen möchten, dann müssen Sie explizit für dieses Element gesetzt werden.

Beispiel:

```

DEFINE DATA LOCAL
1  VARI  (A4)  INIT <'1234'>                                /*      Output
END-DEFINE                                                /*      Produced
*
PRINT NOHDR          'Text'          '='      VARI          /*      Text 1234
PRINT NOHDR (PM=I)   'Text'          '='      VARI          /*      Text 4321
PRINT NOHDR          'Text' (PM=I)   '='      VARI (PM=I)   /*      txeT 4321
PRINT NOHDR          'Text' (PM=I)   '='      VARI          /*      txeT 1234
END

```

Welche Parameter Sie verwenden können, sowie Beschreibungen der einzelnen Parameter entnehmen Sie bitte dem Kapitel **Session-Parameter** des *Natural Referenzhandbuchs*.

Feldpositionierung, Text, Attributzuweisung

$$\left\{ \left[\begin{array}{l} nX \\ nT \\ / \end{array} \right] \dots \left[[='] \begin{array}{l} 'text' [(attributes)] \\ 'c'(n) [(attributes)] \\ operand1 [(parameters)] \end{array} \right] \right\} \dots$$

Feldpositionierung

nX

Mit dieser Notation können Sie zwischen den auszugebenden Werten *n* Leerstellen einfügen. *n* darf nicht 0 sein. Beispiel:

Anmerkung:

n darf nicht 0 sein. (gilt nur bei Großrechner)

Beispiel:

```
PRINT NAME 5X SALARY
```

nT

Mit dieser Notation können Sie Tabulatoren setzen, d.h. die Ausgabe eines Wertes beginnt ab Spalte *n*. Wird ein Tabulator gesetzt, dessen Position bereits durch einen anderen ausgegeben Wert besetzt ist, erfolgt ein Zeilenvorschub.

```
PRINT 25T NAME 50T SALARY
```

(In obigem Beispiel wird NAME ab Spalte 25 ausgegeben und SALARY ab Spalte 50.)

/

Mit einem Schrägstrich bewirken Sie zwischen zwei Feldern oder Textelementen einen Zeilenvorschub. Beispiel:

```
PRINT NAME / SALARY
```

Text/Attributzuweisung

'text'

In Apostrophen angegebener 'text' wird als Text ausgegeben. Beispiel:

```
PRINT 'EMPLOYEE' NAME 'MARITAL/STATUS' MAR-STAT
```

'c'(n)

Wie 'text'. Ausnahme: das Zeichen *c* wird *n*-mal ausgegeben. Beispiel:

```
PRINT '*' (5) '=' NAME
```

'='

Ein Gleichheitszeichen in Apostrophen unmittelbar vor einem Feld bewirkt, daß unmittelbar vor dem Feldwert der Name des Feldes ausgegeben wird. Beispiel:

```
PRINT '=' NAME
```

attributes

Dient dazu, den ausgegebenen Feldern/Textelementen Anzeige- und Farbattribute zuzuordnen. *Attributes* können sein:



- ① Anzeigeattribute (siehe Session-Parameter AD im *Natural Referenzhandbuch*).
- ② Farbattribute (siehe Session-Parameter CD im *Natural Referenzhandbuch*).

operand1

Als *operand1* geben Sie das Feld an, das ausgegeben werden soll.

parameters

Unmittelbar nach *operand1* können Sie in Klammern einen oder mehrere Parameter angeben. Diese Parameter haben dann für das jeweilige Feld Gültigkeit statt der betreffenden mit einem GLOBALS-Kommando, SET GLOBALS- oder FORMAT-Statement gesetzten Parameter. Werden mehrere Parameter angegeben, müssen sie jeweils durch ein oder mehrere Leerzeichen voneinander getrennt werden. Die Angabe eines Parameters darf sich nicht über zwei Sourcecode-Zeilen erstrecken.

Beispiel

```

/* EXAMPLE 'PRTEX1': PRINT
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
  2 JOB-TITLE
  2 ADDRESS-LINE (2)
END-DEFINE
LIMIT 1
READ EMPLOY-VIEW BY CITY
/*****
WRITE NOTITLE 'EXAMPLE 1:'
      // 'RESULT OF WRITE STATEMENT:'
WRITE      / NAME ', ' FIRST-NAME ':' JOB-TITLE '**' (30)
WRITE      / 'RESULT OF PRINT STATEMENT:'
PRINT     / NAME ', ' FIRST-NAME ':' JOB-TITLE '**' (30)
/*****
WRITE      // 'EXAMPLE 2:'
      // 'RESULT OF WRITE STATEMENT:'
WRITE      / NAME 60X ADDRESS-LINE (1:2)
WRITE      / 'RESULT OF PRINT STATEMENT:'
PRINT     / NAME 60X ADDRESS-LINE (1:2)
/*****
END-READ
END

```

EXAMPLE 1:

RESULT OF WRITE STATEMENT:

```
SENKO                , WILLIE                : PROGRAMMER
*****
```

RESULT OF PRINT STATEMENT:

```
SENKO , WILLIE : PROGRAMMER *****
```

EXAMPLE 2:

RESULT OF WRITE STATEMENT:

```
SENKO
2200 COLUMBIA PIKE   #914
```

RESULT OF PRINT STATEMENT:

```
SENKO                2200 COLUMBIA
PIKE #914
```

PROCESS

Anmerkung:

Dieses Statement ist nur mit Entire System Server verfügbar.

PROCESS *view-name* **USING** *operand1=operand2* **GIVING** *operand3...*

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A N P B	ja	nein
Operand2	C S	A N P B	ja	nein
Operand3	S	A N P B	ja	nein

Funktion

Das Statement PROCESS wird in Verbindung mit Entire System Server eingesetzt. Mit Entire System Server können Sie auf verschiedene Funktionen des Betriebssystems zugreifen, zum Beispiel: Lesen/Beschreiben von Dateien, VTOC/Catalog-Management, JES-Queues, usw.

Nähere Informationen zum PROCESS-Statement und seinen Klauseln finden Sie in der *Entire System Server-Dokumentation*.

USING

Mit dieser Klausel können Parameter an den Entire System Server-Prozessor übergeben werden, indem einem Feld (*operand1*) eines unter Entire System Server definierten Views ein Wert (*operand2*) zugewiesen wird. View-Beschreibung siehe *Entire System Server-Dokumentation*.

Anmerkung:

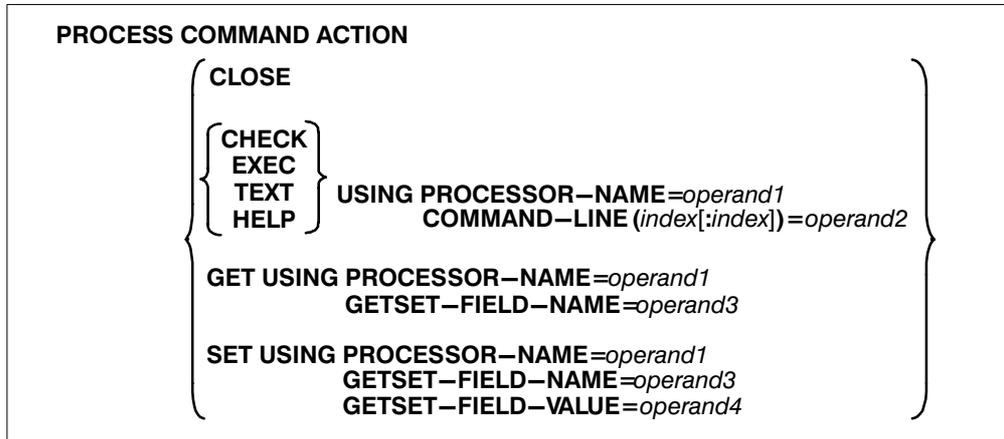
Mehrfache Angaben von "operand1=operand2" müssen entweder mit dem Input-Delimiterzeichen (wie mit dem Session-Parameter ID definiert) oder mit einem Komma voneinander getrennt werden. Ein Komma darf hierzu allerdings nicht verwendet werden, falls das Komma als Dezimalkomma (mit dem Session-Parameter DC) definiert ist.

GIVING

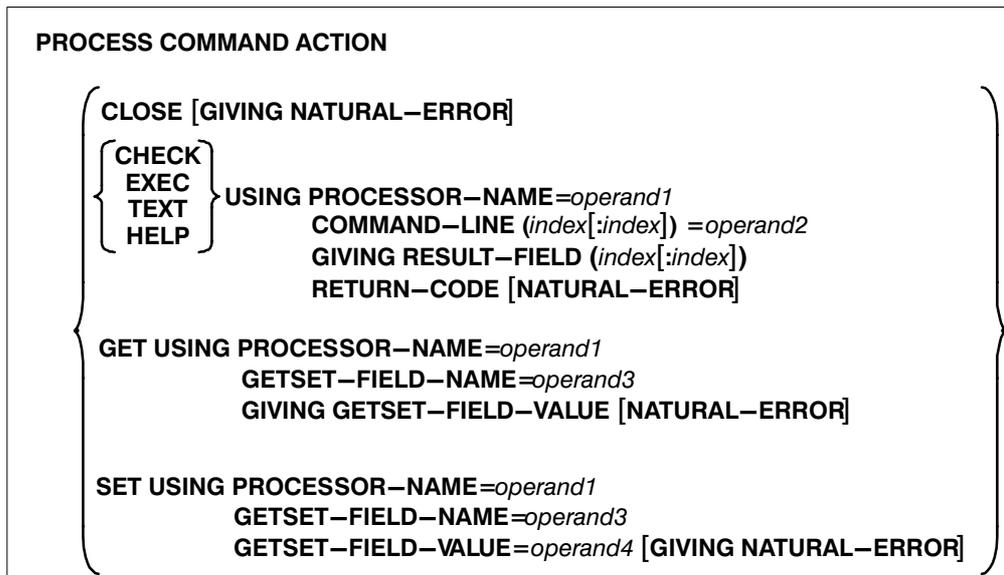
Mit der GIVING-Klausel können Sie Felder (*operand3*) angeben, an die vom Entire System Server-Prozessor Werte zurückgegeben werden. Jedes dieser Felder muß in einem von Entire System Server benutzten View definiert sein.

PROCESS COMMAND

Structured-Mode-Syntax



Reporting-Mode-Syntax



Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	nein	nein
Operand2	C S A G	A N	nein	nein
Operand3	C S	A N	nein	nein
Operand4	C S	A N P I	nein	nein

Funktion

Sobald ein Kommando-Prozessor mit der Natural-Utility SYSNCP erstellt worden ist, kann er von einem Natural-Programm mit dem Statement PROCESS COMMAND aufgerufen werden.

Näheres zur Erstellung eines Natural-Kommando-Prozessors finden Sie unter **Command Processor Maintenance** in Ihrem *Natural User's Guide* bzw. in Ihrer *Natural SYSNCP Utility Documentation*.

Anmerkung:

Das Wort "COMMAND" im Statement PROCESS COMMAND ist eigentlich der Name eines Views. Der Name des verwendeten Views muß nicht unbedingt "COMMAND" sein; aber wir empfehlen die Verwendung von "COMMAND", da ein DDM dieses Namens existiert. Dieses DDM muß im DEFINE DATA-Statement referenziert werden, zum Beispiel "COMMAND VIEW OF COMMAND". Inhalt und Verwendung dieses DDMs sind auf Seite 496 beschrieben.

CLOSE

CLOSE beendet den Einsatz des Kommando-Prozessors und gibt den Kommando-Prozessor-Puffer wieder frei. Wenn der Kommando-Prozessor während einer Session benutzt und nicht mit CLOSE freigegeben wird, enthält Ihr Thread einen Puffer namens NCPWORK. Die Parameter dieses Puffers können Sie mit dem Systemkommando BUS (das im *Natural Benutzerhandbuch für Großrechner* beschrieben ist) auswerten.

Der Puffer wird vom Laufzeit-Teil des Kommando-Prozessors benötigt; er kann mit dem Statement PROCESS COMMAND ACTION CLOSE wieder freigegeben werden. Wenn auf dieses Statement ein anderes PROCESS COMMAND-Statement folgt, wird der Kommando-Prozessor-Puffer erneut geöffnet.

CHECK

CHECK dient als Vorsichtsmaßnahme, um herauszufinden, ob ein Kommando mit dem Statement PROCESS COMMAND EXEC ausführbar ist: für einen bestimmten Prozessor-Namen erfolgt zur Laufzeit eine Prüfung in zwei Schritten:

- Es wird geprüft, ob der Prozessor in der aktuellen Library oder einer ihrer Steplibs vorhanden ist.
- Es wird geprüft, ob der Inhalt der Kommandozeile COMMAND-LINE (1) akzeptabel ist.

Außerdem werden die Laufzeit-Aktionen “R”, “M” und “1–9” in RESULT-FIELD (1:9) geschrieben.

Wenn Sie das Feld NATURAL-ERROR im View oder in der GIVING-Klausel angeben, wird der Fehlercode in diesem Feld ausgegeben. Wird dieses Feld nicht angegeben und die Kommandoanalyse stößt auf einen Fehler, dann erzeugt Natural einen Systemfehler.

Anmerkung:

Da die Funktion der CHECK-Option auch als Teil der EXEC-Option ausgeführt wird (siehe unten), ist es nicht nötig, CHECK vor jeder EXEC-Option zu verwenden.

EXEC

EXEC funktioniert genau wie CHECK und bewirkt zusätzlich, daß die mit dem Runtime Action Editor angegebenen Laufzeit-Aktionen ausgeführt werden.

Es wird nur COMMAND-LINE (1) benötigt. Sie können bis zu 9 Ausprägungen von RESULT-FIELD verwenden (im Hinblick auf optimale Verarbeitungszeit sollten Sie jedoch nur so viele Ausprägungen verwenden wie Sie tatsächlich benötigen).

Anmerkung:

EXEC ist die einzige Option, mit der das gerade aktive Programm verlassen werden kann. Dies ist der Fall, wenn die Laufzeit-Aktion ein FETCH- oder STOP-Statement enthält.

HELP

Mit HELP erhalten Sie eine Liste aller gültigen Schlüsselwörter, Synonyme und Funktionen, um beispielsweise Online-Hilfe-Fenster zu erzeugen. Die Liste ist in dem Feld bzw. den Feldern von RESULT-FIELD enthalten. Die Art der erhaltenen Hilfe hängt vom Inhalt der Kommandozeilen ab: COMMAND-LINE (1) muß die Suchkriterien enthalten; COMMAND-LINE (2) (falls angegeben) muß den Startwert oder einen Suchwert enthalten; COMMAND-LINE (3) (falls angegeben) muß einen Startwert enthalten.

Anmerkung:

Im Hinblick auf optimale Verarbeitungszeit sollte das Feld RESULT-FIELD nicht mehr Ausprägungen haben als Zeilen auf dem Schirm angezeigt werden sollen. Mindestens eine Ausprägung ist erforderlich.

HELP für Schlüsselwörter

Diese Option liefert eine alphabetisch sortierte Liste von Schlüsselwörtern bzw. Synonymen und ihren IKNs.

Kommandozeile	Inhalt
1	<p>Muß mit Indikator “K” (für “Keyword”) anfangen.</p> <p>Die Typen der gewünschten Schlüsselwörter:</p> <ul style="list-style-type: none"> * Schlüsselwörter aller Typen 1 Schlüsselwörter vom Typ “1” 2 Schlüsselwörter vom Typ “2” 3 Schlüsselwörter vom Typ “3” P Schlüsselwörter vom Typ “P” (Parameter) <p>Optionen:</p> <ul style="list-style-type: none"> I Gibt zusätzlich zu Schlüsselwörtern die IKN zurück. T Zeigt Schlüsselwort teilweise in Großbuchstaben (um mögliche Abkürzung zu zeigen). S Gibt zusätzlich zu Schlüsselwörtern Synonyme zurück. X Gibt nur Synonyme der angegebenen Schlüsselwörter zurück. A Interne Schlüsselwörter werden auch zurückgegeben. + Suche schließt Startwert nicht mit ein.
2	<p>Startwert für Schlüsselwort-Suche (optional).</p> <p>Standardmäßig beginnt die Suche ab dem Startwert. Wenn Sie jedoch Option “+” angeben, schließt die Suche den Startwert selbst nicht mit ein, sondern beginnt ab dem nächsthöheren Wert.</p>

Im Feld RESULT-FIELD (1:n) erhalten Sie die angegebene Liste.

Beispiel:

Command Line 1: K*X Gibt alle Synonyme aller Schlüsselworttypen zurück.

Command Line 1: K123S Gibt alle Schlüsselwörter vom Typ 1, 2 und 3 einschließlich ihrer Synonyme zurück.

HELP für Synonyme

Für eine bestimmte IKN liefert diese Option das ursprüngliche Schlüsselwort sowie alle Synonyme.

Kommandozeile	Inhalt
1	Muß mit Indikator "S" anfangen. Option: T Zeigt Schlüsselwort teilweise in Großbuchstaben (um mögliche Abkürzung zu zeigen).
2	IKN (Internal Keyword Number = Interne Schlüsselwort-Nummer) des Schlüsselworts im Format N4.

Im Feld RESULT-FIELD (1) erhalten Sie das Schlüsselwort selbst. In den Feldern RESULT-FIELD (2:n) erhalten Sie die Synonyme des Schlüsselworts.

Beispiel:

Input:

```
Command Line 1:  S
Command Line 2:  1003
```

Output:

```
Result-Field 1:  Edit
Result-Field 2:  Maintain
Result-Field 3:  Modify
```

HELP für globale Funktionen

Diese Option liefert eine Liste aller globalen Funktionen.

Kommandozeile	Inhalt
1	<p>Muß mit Indikator "G" anfangen.</p> <p>Optionen:</p> <p>I Die Interne Funktionsnummer (IFN) wird auch zurückgegeben. T Zeigt Schlüsselwort teilweise in Großbuchstaben (um mögliche Abkürzung zu zeigen). S Die Schlüsselwörter werden in RESULT-FIELD in Spaltenform ausgegeben. A Interne Schlüsselwörter werden auch zurückgegeben. 1 Nur Funktionen, die das angegebene Schlüsselwort vom Typ 1 enthalten, werden zurückgegeben. 2 Nur Funktionen, die das angegebene Schlüsselwort vom Typ 2 enthalten, werden zurückgegeben. 3 Nur Funktionen, die das angegebene Schlüsselwort vom Typ 3 enthalten, werden zurückgegeben. + Suche schließt Startwert nicht mit ein.</p>
2	<p>Startwert für Suche nach globalen Funktionen. Schlüsselwörter müssen in der Reihenfolge "123" angegeben werden.</p> <p>Standardmäßig beginnt die Suche ab dem Startwert. Wenn Sie jedoch Option "+" angeben, schließt die Suche den Startwert selbst nicht mit ein, sondern beginnt ab dem nächsthöheren Wert.</p>
3	Muß leer sein.
4	<p>Wenn Sie nur nach globalen Funktionen eines bestimmten Schlüsselworts suchen möchten, geben Sie hier das betreffende Schlüsselwort an.</p> <p>Gleichzeitig müssen Sie den Schlüsselwort-Typ (1, 2 oder 3) als Option (siehe oben) angeben.</p>

Im Feld RESULT-FIELD (1:n) erhalten Sie die angegebene Liste.

Beispiel:

Input:

```
Command Line 1:  G
Command Line 2:  ADD
```

Output:

```
Result-Field 1:  ADD CUSTOMER
Result-Field 2:  ADD FILE
Result-Field 3:  ADD USER
```

HELP für lokale Funktionen

Diese Option liefert eine Liste aller lokalen Funktionen für einen bestimmten Platz.

Kommandozeile	Inhalt
1	<p>Muß mit Indikator "L" anfangen.</p> <p>Optionen:</p> <p>I Die Interne Funktionsnummer (IFN) wird auch zurückgegeben.</p> <p>T Zeigt Schlüsselwort teilweise in Großbuchstaben (um mögliche Abkürzung zu zeigen).</p> <p>S Die Schlüsselwörter werden in RESULT-FIELD in Spaltenform ausgegeben.</p> <p>A Interne Schlüsselwörter werden auch zurückgegeben.</p> <p>1 Nur Funktionen, die das angegebene Schlüsselwort vom Typ 1 enthalten, werden zurückgegeben.</p> <p>2 Nur Funktionen, die das angegebene Schlüsselwort vom Typ 2 enthalten, werden zurückgegeben.</p> <p>3 Nur Funktionen, die das angegebene Schlüsselwort vom Typ 3 enthalten, werden zurückgegeben.</p> <p>C Nur Funktionen, die für den aktuellen Platz definiert sind, werden zurückgegeben (Kommandozeile 3 wird ignoriert).</p> <p>F Ruft "rekursive" Liste lokaler Funktionen auf, d.h. alle lokalen Kommandos, die zum aktuellen/angegebenen Platz führen, werden zurückgegeben.</p>
2	<p>Startwert für Suche nach lokalen Funktionen (optional).</p> <p>Schlüsselwörter müssen in der Reihenfolge "123" angegeben werden.</p>
3	<p>Der Platz, für den die Liste gewünscht wird.</p> <p>Schlüsselwörter müssen in der Reihenfolge "123" angegeben werden.</p> <p>Wenn kein Platz angegeben wird, wird der aktuelle Platz des Kommando-Prozessors genommen.</p>
4	<p>Schlüsselwort-Einschränkung (optional):</p> <p>Wenn Sie ein Schlüsselwort oder eine IKN mit Format N4 angeben, werden nur Funktionen mit diesem Schlüsselwort zurückgegeben.</p>

Im Feld RESULT-FIELD (1:n) erhalten Sie die angegebene Liste.

HELP für IKN

Für eine bestimmte Interne Schlüsselwortnummer (IKN) liefert diese Option das ursprüngliche Schlüsselwort.

Kommandozeile	Inhalt
1	Muß mit "IKN" anfangen. Optionen: A Das interne Schlüsselwort wird gezeigt. T Zeigt Schlüsselwort teilweise in Großbuchstaben (um mögliche Abkürzung zu zeigen).
2	Die zu übersetzende IKN im Format N4.

Im Feld RESULT-FIELD (1) erhalten Sie das Schlüsselwort.

Beispiel:

Input:

Command Line 1: IKN
Command Line 2: 0000002002

Output:

Result-Field 1: CUSTOMER

HELP für IFN

Für eine bestimmte Interne Funktionsnummer (IFN) liefert diese Option die Schlüsselwörter einer Funktion.

Kommandozeile	Inhalt
1	Muß mit "IFN" anfangen. Option: A Funktionen mit internen Schlüsselwörtern werden nicht unterdrückt.
2	Die zu übersetzende IFN im Format N10.
3	Weitere Optionen: S Gibt die zu der IFN gehörigen Schlüsselwörter in RESULT-FIELD (1:3) zurück. T Zeigt Schlüsselwörter teilweise in Großbuchstaben (um mögliche Abkürzungen zu zeigen). L Die IFN wird zurückgegeben, wenn die IFN als Platz verwendet wird. C Die IFN wird zurückgegeben, wenn die IFN als Kommando verwendet wird.

Im Feld RESULT-FIELD (1) erhalten Sie die Funktion. Wenn Sie Option "S" verwenden, erhalten Sie die Funktion in RETURN-FIELD (1:3).

Beispiel:

Input:
Command Line 1: IFN
Command Line 2: 0001048578

Output:
Result-Field 1: DISPLAY INVOICE

TEXT

Mit der Option TEXT erhalten Sie allgemeine Informationen über den Prozessor sowie Text zu einem Schlüsselwort bzw. einer Funktion. Der Text ist derselbe, der bei der Definition eines Kommando-Prozessors mit der SYSNCP-Utility im Keyword Editor oder Action Editor eingegeben wurde.

Anmerkung:

Um auf Text zu Schlüsselwörtern oder Funktionen zugreifen zu können, muß im Feld "Catalog user texts" in der Processor Header Maintenance (Schirm 3) der SYSNCP-Utility ein "Y" eingetragen sein.

TEXT für allgemeine Informationen

Bei *allgemeinen Informationen* muß COMMAND-LINE (*), d.h. alle Kommandozeilen, leer sein. In den bis zu 9 Feldern von RESULT-FIELD erhalten Sie folgende Informationen:

RESULT-FIELD	Inhalt	Format
1	Header 1 for User Text (Kopfzeile 1 für Benutzertext)	Text (A40)
2	Header 2 for User Text (Kopfzeile 2 für Benutzertext)	Text (A40)
3	"First Entry used as"-Eintrag (Erster Eintrag benutzt als)	Text (A16)
4	"Second Entry used as"-Eintrag (Zweiter Eintrag benutzt als)	Text (A16)
5	"Third Entry used as"-Eintrag (Dritter Eintrag benutzt als)	Text (A16)
6	Anzahl der Eintrag-1-Schlüsselwörter.	Numerisch (N3)
7	Anzahl der Eintrag-2-Schlüsselwörter.	Numerisch (N3)
8	Anzahl der Eintrag-3-Schlüsselwörter.	Numerisch (N3)
9	Anzahl katalogisierter Funktionen.	Numerisch (N7)

TEXT für Schlüsselwort-Informationen

Bei *Schlüsselwort-Informationen* muß COMMAND-LINE (1) das betreffende Schlüsselwort enthalten; COMMAND-LINE (2) kann bei Bedarf den Schlüsselwort-Typ (1, 2, 3 oder P) enthalten; COMMAND-LINE (3:6) muß leer sein.

RESULT-FIELD	Inhalt	Format
1	Schlüsselwort-Kommentartext	Text (A40)
2	Schlüsselwort in voller Länge	Text (A16)
3	Schlüsselwort eindeutig abgekürzt	Text (A16)
4	“Keyword used as”-Eintrag (Schlüsselwort benutzt als)	Text (A16)
5	Interne Schlüsselwortnummer (IKN)	Numerisch (N4)
6	Mindestlänge des Schlüsselworts	Numerisch (N2)
7	Maximallänge des Schlüsselworts	Numerisch (N2)
8	Schlüsselwort-Typ (1, 2, 3, 1S, 2S, 3S, P)	Text (A2)

TEXT für Funktionsinformationen

Bei *Funktionsinformationen* muß COMMAND-LINE (1:3) die Schlüsselwörter enthalten, die den gewünschten Platz bestimmen. COMMAND-LINE (4:6) muß die Schlüsselwörter enthalten, die die gewünschte Funktion bestimmen. Falls beispielsweise Informationen über das globale Kommando ADD USER gewünscht werden, müssen die Kommandozeilen 1, 2, 3 und 6 leer sein, in Kommandozeile 4 muß “ADD” stehen und in Kommandozeile 5 “USER”.

RESULT-FIELD	Inhalt	Format
1	Text wie mit Option “T” in Laufzeit-Aktion definiert.	Text (A40)
2	Interne Funktionsnummer (IFN) des angegebenen Platzes.	Numerisch (N10)
3	Interne Funktionsnummer (IFN) der angegebenen Funktion.	Numerisch (N10)

GET

Die Option GET dient dazu, interne Kommando-Prozessor-Informationen und die aktuellen Kommando-Prozessor-Einstellungen aus dem dynamisch zugewiesenen NCPWORK-Puffer zu lesen. Folgende Felder werden verwendet:

Feldname	Inhalt
GETSET-FIELD-NAME (A32)	Der Name der Variablen, die gelesen werden soll.
GETSET-FIELD-VALUE (A32)	Der Wert der angegebenen Variablen nach der Ausführung von PROCESS COMMAND ACTION GET.

Eine Liste der möglichen Werte von GETSET-FIELD-NAME finden Sie auf der nächsten Seite.

SET

Die Option SET dient dazu, interne Einstellungen des Kommando-Prozessors im NCPWORK-Puffer zu ändern.

Feldname	Inhalt
GETSET-FIELD-NAME (A32)	Der Name der Variablen, die geändert werden soll.
GETSET-FIELD-VALUE (A32)	Der Wert, der der angegebenen Variablen zugewiesen werden soll.

Eine Liste der möglichen Werte von GETSET-FIELD-NAME finden Sie auf der nächsten Seite.

Die möglichen Werte von GETSET-FIELD-NAME sind:

Feldname	Format	G/S*	Inhalt
NAME	A8	G	Aktueller Prozessor-Name.
LIBRARY	A8	G	Geladen aus Library.
FNR	N10	G	Geladen aus Datei.
DBID	N10	G	Geladen aus Datenbank.
TIMESTMP	A8	G	Zeitstempel des aktuellen Prozessors.
COUNTER	N10	G	Zugriffszähler.
BUFFER-LENGTH	N10	G	Für NCPWORK allozierte Bytes.
C-DELIMITER	A1	G/S	Delimiter für mehrere Kommandos.
DATA-DELIMITER	A1	G	Präfix für Daten.
PF-KEY	A1	G/S	PF-Taste kann Kommando sein (Y/N).
UPPER-CASE	A1	G	Schlüsselwörter in Großbuchstaben (Y/N).
UQ-KEYWORDS	A1	G	Eindeutige Schlüsselwörter (Y/N).
IMPLICIT-KEYWORD	A1	G/S	Identifiziert impliziten Schlüsselwort-Eintrag.
MIN-LEN	N10	G	Schlüsselwort-Mindestlänge.
MAX-LEN	N10	G	Schlüsselwort-Maximallänge.
KEYWORD-SEQ	A8	G/S	Schlüsselwort-Reihenfolge.
ALT-KEYWORD-SEQ	A8	G/S	Alternative Schlüsselwort-Reihenfolge.
USER-SEQUENCE	A1	G	Benutzer darf KEYWORD-SEQ überschreiben (Y/N).
CURR-LOCATION	N10	G/S	Aktueller Platz (IFN).
CURR-IKN1	N10	G/S	IKN1 des aktuellen Platzes.
CURR-IKN2	N10	G/S	IKN2 des aktuellen Platzes.
CURR-IKN3	N10	G/S	IKN3 des aktuellen Platzes.
CHECK-LOCATION	N10	G	Letzter geprüfter Platz (IFN).
CHECK-IKN1	N10	G	IKN1 von CHECK-LOCATION.
CHECK-IKN2	N10	G	IKN2 von CHECK-LOCATION.
CHECK-IKN3	N10	G	IKN3 von CHECK-LOCATION.
TOP-IKN1	N10	G	IKN1 des obersten Schlüsselworts.
TOP-IKN2	N10	G	IKN2 des obersten Schlüsselworts.
TOP-IKN3	N10	G	IKN3 des obersten Schlüsselworts.
KEY1-TOTAL	N10	G	Anzahl von Schlüsselwörtern vom Typ 1.

Feldname	Format	G/S*	Inhalt
KEY2-TOTAL	N10	G	Anzahl von Schlüsselwörtern vom Typ 2.
KEY3-TOTAL	N10	G	Anzahl von Schlüsselwörtern vom Typ 3.
FUNCTIONS-TOTAL	N10	G	Anzahl der katalogisierten Funktionen.
LOCAL-GLOBAL-SEQ	A8	G/S	Lokale/globale Funktionsvalidierung.
ERROR-HANDLER	A8	G/S	Allgemeines Fehlerprogramm.
SECURITY	A1	G	Natural Security installiert (Y/N).
SEC-PREFETCH	A1	G	Natural Security-Daten sollen gelesen werden (Y/N) bzw. wurden gelesen (D = done).
PREFIX1	A1	G	Entspricht dem Feld "Prefix Character 1" des Schirms "Processor Header Maintenance 2".
PREFIX2	A1	G	Entspricht dem Feld "Prefix Character 2" des Schirms "Processor Header Maintenance 2".
HEX1	A1	G	Entspricht dem Feld "Hex. Replacement 1" des Schirms "Processor Header Maintenance 2".
HEX2	A1	G	Entspricht dem Feld "Hex. Replacement 2" des Schirms "Processor Header Maintenance 2".
DYNAMIC	A32	G	Dynamischer Teil (:n:) der letzten Fehlermeldung.
LAST	–	G	Letztes oben auf dem Stack als Daten abgelegtes Kommando.
LAST-ALL	–	G	Letzte oben auf dem Stack als Daten abgelegte Kommandos.
LAST-COM	–	G	Letztes in *COM gestelltes Kommando.
MULTI	–	G	Legt das letzte von mehreren Kommandos als Daten oben auf dem Stack ab.
MULTI-COM	–	G	Legt das letzte von mehreren Kommandos in der Systemvariablen *COM ab.

* **G** = Kann mit der GET-Option verwendet werden.

S = Kann mit der SET-Option verwendet werden.

USING-Klausel

Die Inhalte der Felder in der USING-Klausel bestimmen zum Beispiel den Prozessor-Namen und die Kommandozeile.

In der USING-Klausel werden die Felder angegeben, die an den Kommando-Prozessor übergeben werden.

Option	Feldname			
	PROCESSOR-NAME	COMMAND-LINE	GETSET-FIELD-NAME	GETSET-FIELD-VALUE
CLOSE				
CHECK	X	X		
EXEC	X	X		
TEXT	X	X		
HELP	X	X		
GET	X		X	
SET	X		X	X

Legende: X = Feld muß angegeben werden.

GIVING-Klausel

Diese Klausel kann nur im Reporting Mode verwendet werden.

In der GIVING-Klausel werden die Felder angegeben, die vom Kommando-Prozessor gefüllt werden, wenn eine Option verarbeitet wird.

Option	Feldname			
	NATURAL- ERROR	RETURN- CODE	RESULT- FIELD	GETSET- FIELD-VALUE
CLOSE	R			
CHECK	R	X	X	
EXEC	R	X	X	
TEXT	R	X	X	
HELP	R	X	X	
GET	R			X
SET	R			

Legende: X = Feld muß angegeben werden.
R = Feld sollte angegeben werden (muß aber nicht).

Anmerkung:

Die GIVING-Klausel kann im Structured Mode nicht verwendet werden, da es eine implizite GIVING-Klausel gibt, die sich aus allen im DEFINE DATA-Statement angegebenen Feldern zusammensetzt, die für gewöhnlich in der GIVING-Klausel des Reporting Modes referenziert werden.

Das bedeutet, daß im Structured Mode alle in obiger Tabelle mit "X" markierten Felder im DEFINE DATA-Statement definiert sein müssen.

Das DDM “COMMAND”

Das DDM “COMMAND” wurde speziell zur Verwendung mit dem PROCESS COMMAND-Statement erstellt:

DB:		1 File:		1 - COMMAND		Default Sequence: ?	
TYL	DB	NAME	F	LENG	S	D	REMARKS
	1	AA	PROCESSOR-NAME	A	8	N D DE	USING
M	1	AB	COMMAND-LINE	A	80	N D MU/DE	USING
	1	AF	GETSET-FIELD-NAME	A	32	N D DE	USING
	1	BA	NATURAL-ERROR	N	4.0	N	GIVING
	1	BB	RETURN-CODE	A	4	N	GIVING
M	1	BC	RESULT-FIELD	A	80	N MU	GIVING
	1	BD	GETSET-FIELD-VALUE	A	32	N D	USING; GIVING
***** DDM OUTPUT TERMINATED *****							

Anmerkung:

Um mögliche Kompilierungs- bzw. Laufzeitfehler zu vermeiden, prüfen Sie bitte, ob das DDM “COMMAND” als Typ “C” (Feld “DDM Type” auf dem SYSDDM-Menü) katalogisiert ist, bevor Sie es verwenden. (Falls Sie es neu katalogisieren, werden hierbei etwaige DBID/FNR-Angaben in SYSDDM ignoriert.)

Das DDM “COMMAND” enthält folgende Felder:

DDM-Feld	Erklärung
PROCESSOR-NAME	Der Name des Kommando-Prozessors, für den das PROCESS COMMAND-Statement ausgeführt wird. Der Prozessor muß katalogisiert sein.
COMMAND-LINE	Die Kommandozeile, die vom Kommando-Prozessor verarbeitet werden soll (Optionen CHECK, EXEC) bzw. das Schlüsselwort/Kommando, für das Benutzertext oder Hilfe-Text an das Programm zurückgegeben werden soll (Optionen TEXT, HELP). Bitte beachten Sie, daß dieses Feld aus mehr als einer Zeile bestehen kann.
GETSET-FIELD-NAME	Dieses Feld wird mit den Optionen GET und SET verwendet und dient zur Angabe des Namens der Konstanten/Variablen, die gelesen (GET) oder geschrieben (SET) werden soll.
RETURN-CODE	Dieses Feld enthält den Return Code einer Aktion aufgrund der Option EXEC oder CHECK, wie in einer Laufzeit-Aktion angegeben (siehe SYSNCP-Utility).
NATURAL-ERROR	Dieses Feld wird mit allen Optionen verwendet. Wenn es im DEFINE DATA angegeben wird, enthält es den Natural-Fehlercode für den Kommando-Prozessor. Wenn das Feld fehlt, ist die normale Natural-Fehlerbehandlung aktiv.
RESULT-FIELD	Dieses Feld enthält Informationen, die aus der Verwendung verschiedener Optionen, wie in einer Laufzeit-Aktion angegeben, resultieren (siehe den Abschnitt Runtime Actions in der <i>Natural SYSNCP Utility Documentation</i>). Bitte beachten Sie, daß dieses Feld aus mehr als einer Zeile bestehen kann.
GETSET-FIELD-VALUE	Dieses Feld wird mit den Optionen GET und SET verwendet und enthält den Wert der im Feld GETSET-FIELD-NAME angegebenen Konstanten/Variablen (siehe oben).

Security-Hinweise

Mit Natural Security können Sie die Verwendung bestimmter in einem Kommando-Prozessor definierter Schlüsselwörter und/oder Funktionen einschränken. Schlüsselwörter und Funktionen können für jeden einzelnen Benutzer (oder Gruppen von Benutzern) erlaubt bzw. verboten werden. Nähere Informationen hierzu finden Sie in Ihrer *Natural Security Documentation*.

Beispiel 1

```
/* EXAM-CLS - Example for PROCESS COMMAND ACTION CLOSE (Structured Mode)
/*****
DEFINE DATA LOCAL
    01 COMMAND VIEW OF COMMAND
END-DEFINE
/*
PROCESS COMMAND ACTION CLOSE
/*
DEFINE WINDOW CLS
INPUT WINDOW = 'CLS'
    'NCPWORK has just been released.'
/*
END
```

Beispiel 2

```

/* EXAM-EXS - Example for PROCESS COMMAND ACTION EXEC (Structured Mode)
/*****
DEFINE DATA LOCAL
  01 COMMAND VIEW OF COMMAND
    02 PROCESSOR-NAME
    02 COMMAND-LINE (1)
    02 NATURAL-ERROR
    02 RETURN-CODE
    02 RESULT-FIELD (1)
  01 MSG (A65) INIT <'Please enter a command.'>
END-DEFINE
/*
REPEAT
  INPUT (AD=MIT' ' IP=OFF) WITH TEXT MSG
    'Example for PROCESS COMMAND ACTION EXEC (Structured Mode)' (I)
  / 'Command ==>' COMMAND-LINE (1) (AL=64)
  /*****
  PROCESS COMMAND ACTION EXEC
  USING
    PROCESSOR-NAME = 'DEMO'
    COMMAND-LINE (1) = COMMAND-LINE (1)
  /*****
  COMPRESS 'NATURAL-ERROR =' NATURAL-ERROR TO MSG
END-REPEAT
END

```

Anmerkung:

Weitere Beispielprogramme finden Sie in der Library SYSNCP. Die Namen der Beispielprogramme beginnen alle mit "EXAM".

PROCESS GUI

Anmerkung:

Dieses Statement ist nur unter Windows verfügbar.

PROCESS GUI ACTION <i>action-name</i> WITH $\left\{ \left\{ \begin{array}{c} \textit{operand1} \\ nX \end{array} \right\} \dots \right\} [\textit{GIVING operand2}]$ $\left\{ \textit{PARAMETERS-clause} \right\}$
--

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1*	C S A	A N P I F B D T L G	ja	nein
Operand2	S	N P I	ja	nein

* Welche(s) Struktur/Format tatsächlich möglich ist, hängt jeweils von der auszuführenden Aktion ab.

Funktion

Das Statement PROCESS GUI dient dazu, eine Aktion auszuführen. Mit Aktion ist in diesem Zusammenhang eine Prozedur gemeint, die in ereignis-gesteuerten Anwendungen häufig benötigt wird.

Allgemeine Informationen zu diesen Standardprozeduren finden Sie unter **Event-Driven Programming Techniques** im *Natural User's Guide for Windows*.

Informationen zu den einzelnen verfügbaren Aktionen und ihren Parametern sowie Beispiele finden Sie unter **Executing Standardized Procedures** in der *Natural Dialog Components Documentation for Windows*.

action-name

Als *action-name* geben Sie den Namen der Aktion an, die aufgerufen werden soll.

Übergabe von Parametern an die Aktion

Als *operand1* geben Sie die Parameter an, die an die Aktion übergeben werden sollen. Die Parameter werden in der angegebenen Reihenfolge übergeben.

Für die Aktion “ADD” können Sie Parameter auch namentlich (statt in Angabereihenfolge) übergeben. Hierzu verwenden Sie die *PARAMETERS-clause*:

PARAMETERS-clause

PARAMETERS { *parameter-name = operand1* } ... END-PARAMETERS

Diese Klausel kann nur für die Aktion “ADD” verwendet werden, aber nicht für andere Aktionen.

Wenn die Aktion optionale Parameter hat (d.h. Parameter, die nicht angegeben zu werden brauchen), können Sie die Notation *nX* als Platzhalter für *n* nicht angegebene Parameter verwenden. Zur Zeit sind die einzigen Aktionen, die optionale Parameter haben können, die Methoden und parametrisierten Eigenschaften von ActiveX Controls.

nX

Mit der Notation *nX* können Sie angeben, daß die nächsten *n* Parameter übersprungen werden sollen (z.B. 1X, um den nächsten Parameter zu überspringen, oder 3X, um die nächsten drei Parameter zu überspringen); dies bedeutet, daß für die nächsten *n* Parameter keine Werte an die Aktion übergeben werden. Dies ist nur möglich bei Aktionen, die für ActiveX Controls gelten.

Ein zu überspringender Parameter muß mit dem Schlüsselwort OPTIONAL in der Methode der ActiveX Control definiert werden. OPTIONAL bedeutet, daß ein Wert vom aufrufenden Objekt an einen solchen Parameter übergeben werden kann — aber nicht unbedingt muß.

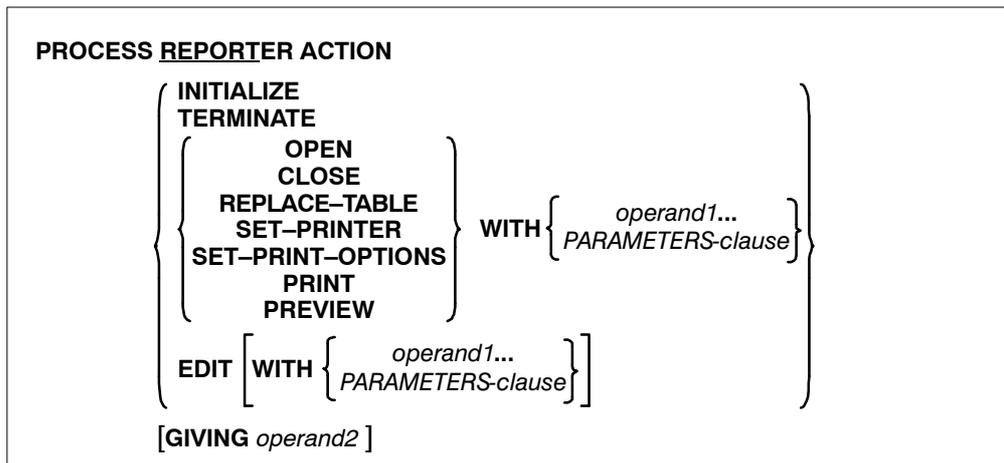
GIVING *operand2*

Als *operand2* können Sie ein Feld angeben, das den Response Code der aufgerufenen Aktion erhält, nachdem die Aktion ausgeführt wurde.

PROCESS REPORTER

Anmerkung:

Dieses Statement ist nur unter Windows und Windows NT verfügbar.



Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A N P I F B D T L	ja	nein
Operand2	S	N P I	ja	nein

Funktion

Das Statement PROCESS REPORTER dient dazu, mit dem Natural-Reporter zu kommunizieren und den Reporter von einem Programm aus anzuweisen, eine bestimmte Aktion auszuführen.

Eine Beschreibung des Reporters finden Sie in der Online-Hilfe vom *Natural Reporter*.

Anmerkung:

Für Aktionen, die sich auf einen bestimmten Report beziehen, können Sie das zweite Schlüsselwort als REPORT abkürzen. Dies dient lediglich zur besseren Lesbarkeit des Programms; Natural unterscheidet nicht zwischen abgekürzter und ausgeschriebener Schreibweise des Schlüsselworts.

Aktionen

Sie können eine der folgenden Aktionen angeben, die vom Reporter ausgeführt werden soll:

- **INITIALIZE** — Diese Aktion initialisiert und lädt den Reporter. Dies muß jeweils die erste Aktion sein, die ausgeführt wird.
- **TERMINATE** — Diese Aktion beendet und entlädt den Reporter. Dies muß jeweils die letzte Aktion sein, die ausgeführt wird.
- **OPEN** — Diese Aktion öffnet einen angegebenen Report und gibt eine Handle zurück, die in nachfolgenden Aktionen zur Identifikation des Reports verwendet werden kann.
- **CLOSE** — Diese Aktion schließt einen angegebenen Report; danach kann die Handle des Reports nicht mehr verwendet werden.
- **REPLACE-TABLE** — Diese Aktion ersetzt den Pfadnamen einer Tabelle.
- **SET-PRINTER** — Diese Aktion wählt einen Drucker aus, auf dem anschließend alle Reports gedruckt werden können. Die Druckmethode für den ausgewählten Drucker muß in NATPARM auf "TTY" gesetzt sein.
- **SET-PRINT-OPTIONS** — Diese Aktion dient zum Setzen von Druckoptionen für einen angegebenen Report.
- **PRINT** — Diese Aktion druckt einen angegebenen Report auf dem zur Zeit ausgewählten Drucker.
- **PREVIEW** — Diese Aktion zeigt einen angegebenen Report an, so wie er (auf dem zur Zeit ausgewählten Drucker) ausgedruckt würde.
- **EDIT** — Falls kein Report angegeben wird, zeigt diese Aktion das Reporter-Hauptfenster. Falls ein Report angegeben wird, zeigt diese Aktion das Reporter-Hauptfenster zusammen mit dem Edit-Fenster des angegebenen Reports.

WITH-Klausel

Als *operand1* geben Sie die Parameter an, die an die Aktion übergeben werden.

Wahlweise können Sie auch die *PARAMETERS-clause* verwenden:

PARAMETERS-clause

```
PARAMETERS {parameter-name = operand1} ... END-PARAMETERS
```

Mit dieser Klausel geben Sie die Parameter namentlich (anstatt als Positionsparameter) an.

Parameter für Aktion OPEN

Für diese Aktion geben Sie als ersten Parameter den Namen (ohne “.rpt”-Erweiterung oder Pfadangabe) des Reports an, der geöffnet werden soll, und als zweiten Parameter das Feld, in das die Handle zurückgegeben werden soll. Format/Länge des ersten Parameters muß mit A8, Format/Länge des zweiten Parameters mit I4 kompatibel sein.

Der Report wird zuerst im Unterverzeichnis RES der Logon-Library, dann im Unterverzeichnis RES jeder Steplib und dann in dem der Umgebungsvariablen NATGUI_BMP zugewiesenen Verzeichnis gesucht.

Anmerkung:

Die Reportdaten werden zuerst bei dem bei Erstellung des Reports angegebenen Pfad (wenn überhaupt vorhanden) gesucht, dann im Verzeichnis, in dem der Report gefunden wurde.

Falls Sie die *PARAMETERS-clause* verwenden, muß der *parameter-name* für den Reportnamen “REPORT-NAME” und für das Handle-Feld “REPORT-ID” sein.

Beispiele:

```
PROCESS REPORT ACTION OPEN WITH 'MYREPORT' #HANDLE
PROCESS REPORT ACTION OPEN WITH
PARAMETERS
  REPORT-NAME = 'MYREPORT'
  REPORT-ID   = #HANDLE
END-PARAMETERS
```

Parameter für Aktion REPLACE-TABLE

Für diese Aktion geben Sie als ersten Parameter die Handle des Reports an, auf den die Aktion angewandt werden soll, als zweiten Parameter die Arbeitsdateinummer und, optional, als dritten Parameter den Tabellennamen. Format/Länge der ersten beiden Parameter muß mit I4, Format/Länge des dritten Parameters mit A8 kompatibel sein.

Falls Sie die *PARAMETERS-clause* verwenden, müssen Sie als *parameter-name* "REPORT-ID", "WORK-FILE" bzw. "TABLE-NAME" verwenden.

Beispiel:

```
PROCESS REPORT ACTION REPLACE-TABLE WITH  
PARAMETERS  
  REPORT-ID = #HANDLE  
  WORK-FILE = 5  
END-PARAMETERS
```

Parameter für Aktion SET-PRINTER

Für diese Aktion geben Sie als *operand1* den logischen Gerätenamen ('LPT1' bis 'LPT31') des auszuwählenden Druckers an. Format/Länge von *operand1* muß mit A8 kompatibel sein.

Falls Sie die *PARAMETERS-clause* verwenden, muß der *parameter-name* "DEVICE-NAME" sein.

Beispiel:

```
PROCESS REPORTER ACTION SET-PRINTER WITH 'LPT1'
```

Parameter für Aktion SET-PRINT-OPTIONS

Für diese Aktion geben Sie als ersten Parameter die Handle des Reports an, auf den die Aktion angewandt werden soll, gefolgt von den Druckeroptionen — die alle optional sind. Falls ein Parameter weggelassen wird, bleibt die entsprechende Option unverändert.

Der 1. Parameter (dessen Format/Länge mit I4 kompatibel sein muß) ist die Handle des Reports, auf den die Aktion angewandt werden soll.

Der 2. Parameter (dessen Format/Länge mit I2 kompatibel sein muß) ist eine der in der Local Data Area NGULKEY1 definierten Papiergröße-Konstanten: Mögliche Werte sind:

- CUSTOM-PAPER (Papierhöhe und -breite werden explizit angegeben)
- LETTER (8,5 x 11 Inches)
- LEGAL (8,5 x 14 Inches)
- EXECUTIVE (7,25 x 10,5 Inches)
- A4 (210 x 297 mm)
- COM-10-ENVELOPE (4,125 x 9,5 Inches)
- DL-ENVELOPE (110 x 220 mm)
- C5-ENVELOPE (162 x 229 mm)
- B5-ENVELOPE (176 x 250 mm)
- MONARCH-ENVELOPE (3,875 x 7,5 Inches).

Der 3. und 4. Parameter (deren Format/Länge mit I2 kompatibel sein muß) sind die Papierbreite bzw. -höhe (in Twips; 1 Twip = 1/1440 Inches). Diese Parameter werden nur bei Papiergröße CUSTOM-PAPER verwendet.

Der 5., 6., 7. und 8. Parameter (deren Format/Länge mit I2 kompatibel sein muß) bestimmen den linken, oberen, rechten bzw. unteren Rand (in Twips).

Der 9. Parameter (der Format L haben muß) bestimmt die Papierausrichtung: TRUE = Querformat (landscape), FALSE = Hochformat (portrait). Dieser Parameter wird nicht bei Papiergröße CUSTOM-PAPER verwendet.

Der 10. Parameter (der Format L haben muß) ist die Schnelldruck-Option (nur Text): TRUE = Graphiken werden unterdrückt, FALSE = keine Unterdrückung.

Der 11. Parameter (der Format L haben muß) bestimmt, ob Datensätze, die nur aus Leerzeichen bestehen, bei der Ausgabe unterdrückt werden sollen: TRUE = Unterdrückung, FALSE = keine Unterdrückung.

Der 12. Parameter (der Format L haben muß) bestimmt, ob nachfolgende Datensätze mit identischen Daten ignoriert werden sollen: TRUE = ignorieren, FALSE = nicht ignorieren.

Der 13. Parameter (der Format L haben muß) bestimmt, ob während des Druckens ein Druckerauswahl-Dialog angezeigt werden soll: TRUE = anzeigen, FALSE = nicht anzeigen.

Der 14. Parameter (dessen Format/Länge mit I2 kompatibel sein muß) ist eine der in der Local Data Area NGULKEY1 definierten Papierquellen-Konstanten. Mögliche Werte sind: AUTOMATIC = automatische Zufuhr, MANUAL = manuelle Zufuhr.

Falls Sie die *PARAMETERS-clause* verwenden, müssen Sie als *parameter-name* REPORT-ID, PAPER-SIZE, PAPER-WIDTH, PAPER-HEIGHT, LEFT-MARGIN, TOP-MARGIN, RIGHT-MARGIN, BOTTOM-MARGIN, LANDSCAPE, FAST-PRINT, SUPPRESS-BLANK-LINES, IGNORE-DUPPLICATES, SHOW-PRINT-DIALOG bzw. PAPER-SOURCE verwenden.

Beispiele:

```
DEFINE DATA LOCAL
  USING 'NGLUKEY1'
END-DEFINE
...
PROCESS REPORT ACTION SET-PRINT-OPTIONS WITH #HANDLE
  A4 0 0 0 0 0 0 FALSE FALSE FALSE FALSE FALSE AUTOMATIC
```

```
DEFINE DATA LOCAL
  USING 'NGLUKEY1'
END-DEFINE
...
PROCESS REPORT ACTION SET-PRINT-OPTIONS WITH PARAMETERS
  REPORT-ID = #HANDLE
  PAPER-SIZE = A4
  PAPER-WIDTH = 0
  PAPER-HEIGHT = 0
  LEFT-MARGIN = 0   TOP-MARGIN = 0
  RIGHT-MARGIN = 0  BOTTOM-MARGIN = 0
  LANDSCAPE = FALSE
  FAST-PRINT = FALSE
  SUPPRESS-BLANK-LINES = FALSE
  IGNORE-DUPPLICATES = FALSE
  SHOW-PRINT-DIALOG = FALSE
  PAPER-SOURCE = AUTOMATIC
END-PARAMETERS
```

Parameter für Aktionen CLOSE, PRINT, PREVIEW, EDIT

Für diese Aktionen geben Sie als *operand1* die Handle des Reports an, auf den die Aktion angewandt werden soll. Format/Länge von *operand1* muß mit I4 kompatibel sein.

Falls Sie die *PARAMETERS-clause* verwenden, muß der *parameter-name* "REPORT-ID" sein.

Beispiele:

```
PROCESS REPORT ACTION PRINT WITH #HANDLE
```

```
PROCESS REPORT ACTION PREVIEW WITH #HANDLE
```

```
PROCESS REPORT ACTION CLOSE WITH #HANDLE
```

```
PROCESS REPORT ACTION EDIT WITH #HANDLE
```

```
PROCESS REPORTER ACTION EDIT
```

GIVING *operand2*

Mit der GIVING-Klausel erhalten Sie den Response Code der aufgerufenen Aktion.

Als *operand2* geben Sie das Feld an, das den Response Code erhalten soll.

Der Response Code wird in Format/Länge I4 zurückgegeben.

Response Code "0" bedeutet, daß die Aktion erfolgreich ausgeführt wurde. Jeder andere Response Code entspricht einer Natural-Systemfehlernummer (NATnnnn).

PROPERTY

```
PROPERTY property-name  
  
      OF [ INTERFACE ] interface-name  
      IS operand  
  
END-PROPERTY
```

Funktion

Das PROPERTY-Statement weist einen Objektdatenvariablen-Operanden als Implementierung für eine Property (Eigenschaft) zu, und zwar außerhalb einer Interface-Definition (Schnittstellen-Definition). Es wird verwendet, wenn die betreffende Interface-Definition von einem Copycode übernommen wird und auf eine klassenspezifische Art und Weise implementiert werden soll.

Es kann nur innerhalb des DEFINE CLASS-Statements und nach den Interface-Definitionen verwendet werden. Die angegebenen Interface- und Property-Namen müssen in den Interface-Definitionen definiert sein.

READ

```

{ READ
  BROWSE } [ ALL ] [ RECORDS ] [ IN ] [ FILE ] view-name
            (operand1)
            [ PASSWORD=operand2 ]
            [ CIPHER=operand3 ]
            [ WITH REPOSITION ]
            [ sequence/range-specification ]
            [ STARTING WITH ISN=operand4 ]
            [ WHERE logical-condition ]
            statement...
END-READ   (nur im Structured-Mode)
[ LOOP ]     (nur im Reporting-Mode)

```

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	N P I B	ja	nein
Operand2	C S	A	ja	nein
Operand3	C S	N	ja	nein
Operand4	C S	N P I B	ja	nein

Verwandte Statements

FIND, HISTOGRAM.

Funktion

Das Statement READ dient dazu, Datensätze von der Datenbank zu lesen. Die Datensätze können in physischer Reihenfolge, in der Reihenfolge der Adabas-ISNs oder in der Reihenfolge der Werte eines Deskriptorfeldes gelesen werden.

Das READ-Statement initiiert eine Verarbeitungsschleife.

Anzahl der Datensätze (*operand1*/ALL)

Sie können die Anzahl der Datensätze, die mit dem READ-Statement gelesen werden sollen, durch Angabe von *operand1* (in Klammern hinter dem Schlüsselwort READ) in Form einer numerischen Konstanten (0 bis 99999999) oder über eine Benutzervariable begrenzen. Zum Beispiel:

```
READ (5) IN EMPLOYEES ...
```

```
MOVE 10 TO CNT(N2)  
READ (CNT) EMPLOYEES ...
```

Das angegebene Limit hat für dieses Statement Vorrang vor einem mit einem LIMIT-Statement gesetzten Limit.

Ist mit dem LT-Parameter ein kleineres Limit gesetzt, so gilt das LT-Limit.

Um zu betonen, daß *alle* Datensätze gelesen werden sollen, können Sie optional das Schlüsselwort ALL angeben.

Anmerkungen:

Wenn Sie eine vierstellige Anzahl von Datensätzen lesen möchten, geben Sie diese mit einer vorangestellten Null an: (0nnnn); denn Natural interpretiert jede vierstellige Zahl in Klammern als Zeilennummer-Referenzierung auf ein Statement.

Operand1 wird zu Beginn des ersten READ-Schleifendurchlaufs ausgewertet. Wird der Wert von operand1 innerhalb der READ-Schleife geändert, hat dies keine Auswirkungen auf die Anzahl der gelesenen Datensätze.

view-name

Als *view-name* geben Sie den Namen eines Views an, der entweder in einem DEFINE DATA-Statement oder in einer programmexternen Global oder Local Data Area definiert ist.

Im *Reporting Mode* darf *view-name* auch der Name eines DDMs sein.

PASSWORD- und CIPHER-Klauseln

*Diese Klauseln gelten nur für Zugriffe auf Adabas- oder VSAM-Datenbanken.
Mit Entire System Server können sie nicht verwendet werden.*

Die PASSWORD-Klausel dient dazu, ein Paßwort anzugeben, um auf Daten einer paßwort-geschützten Datei zugreifen zu können.

Die CIPHER-Klausel dient dazu, einen Cipher-Code (Chiffrierschlüssel) anzugeben, um in chiffrierter Form gespeicherte Daten in entschlüsselter Form zu erhalten.

Weitere Informationen hierzu siehe Statements FIND und PASSW.

WITH REPOSITION

Diese Option ist nur beim Zugriff auf VSAM-Datenbanken möglich.

Mit dieser Option können Sie innerhalb der aktiven READ-Schleife auf einen anderen Startwert für die zu lesenden Datensätze repositionieren. Die Verarbeitung des READ-Statements wird dann unter Verwendung des neuen Startwerts fortgesetzt.

Die Repositionierung wird dadurch ausgelöst, daß der Wert der Systemvariablen *COUNTER auf "0" zurückgesetzt wird; d.h. der neue Startwert wird verwendet, sobald *COUNTER "0" ist.

Beispiel:

```
DEFINE DATA LOCAL
1 MYVIEW VIEW OF ...
  2 NAME
1 #STARTVAL (A20) INIT <'A'>
1 #ATTR      (C)
END-DEFINE
...
SET KEY PF3
...
READ MYVIEW WITH REPOSITION BY NAME = #STARTVAL
  INPUT (IP=OFF AD=0) 'NAME:' NAME /
    'Enter new start value for repositioning:' #STARTVAL (AD=MT CV=#ATTR) /
    'Press PF3 to stop'
  IF *PF-KEY = 'PF3'
    THEN STOP
  END-IF
  IF #ATTR MODIFIED
    THEN RESET *COUNTER
  END-IF
END-READ
...
```

Lesereihenfolge und -umfang (*sequence/range-specification*)

①	[IN] [PHYSICAL]	[ASCENDING DESCENDING VARIABLE <i>operand5</i>]	[SEQUENCE]
②	{ BY WITH } ISN	[= EQ EQUAL TO [STARTING] FROM] <i>operand6</i>	[[THRU ENDING AT]] <i>operand7</i>
③	[IN] [LOGICAL]	[ASCENDING DESCENDING VARIABLE <i>operand5</i>]	[SEQUENCE] { BY WITH } <i>descriptor</i>
		[= EQ EQUAL TO [STARTING] FROM] <i>operand6</i>	[[THRU ENDING AT]] <i>operand7</i>

Optionen ② und ③ sind in Verbindung mit Entire System Server nicht verfügbar.

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand5	S	A	ja	nein
Operand6	C S	A N P I F B D T L	ja	nein
Operand7	C S	A N P I F B D T L	ja	nein

READ IN PHYSICAL SEQUENCE

PHYSICAL SEQUENCE bedeutet, daß die Datensätze in der physischen Reihenfolge, in der sie auf der Datenbank gespeichert sind, gelesen werden.

Bei VSAM-Datenbanken ist READ PHYSICAL nur beim Zugriff auf ESDS und RRDS möglich.

Ist nicht anderes angegeben, gilt standardmäßig IN PHYSICAL SEQUENCE.

READ BY ISN

READ BY ISN ist nur bei Adabas- und VSAM-Datenbanken möglich, und zwar bei VSAM nur für ESDS und RRDS.

READ BY ISN bedeutet, daß die Datensätze in der Reihenfolge der Adabas-ISNs (Interne Satznummern) bzw. VSAM-RBAs (relative Byte-Adressen von ESDS) bzw. -RRNs (relative Datensatznummern von RRDS) gelesen werden.

READ IN LOGICAL SEQUENCE

LOGICAL SEQUENCE bedeutet, daß die Datensätze in der Reihenfolge der Werte eines bestimmten Deskriptorfeldes (*descriptor*) gelesen werden.

Wenn Sie ein Deskriptorfeld angeben, werden die Datensätze in der Wertabfolge dieses Feldes gelesen. Als Feld können Sie einen Deskriptor, Subdeskriptor, Superdeskriptor oder Hyperdeskriptor verwenden, nicht aber einen phonetischen Deskriptor, einen Deskriptor innerhalb einer Periodengruppe oder einen Superdeskriptor, der ein Periodengruppenfeld enthält.

Wenn Sie kein Deskriptorfeld angeben, wird der im verwendeten DDM eingetragene Standarddeskriptor (Feld "Default Sequence") genommen.

Bei **DL/I-Datenbanken** muß der verwendete Deskriptor entweder das Schlüsselfeld eines Wurzelsegments oder ein Sekundärindexfeld sein. Ein verwendetes Sekundärindexfeld muß ebenfalls im PROSEQ-Parameter eines PCB angegeben werden; Natural benutzt diesen PCB und die entsprechende hierarchische Struktur, um die Daten zu lesen.

Wenn Sie READ LOGICAL bei HDAM-Datenbanken verwenden, erhalten Sie keine sinnvollen Ergebnisse, da HDAM-Datenbanken zum Auffinden von Wurzelsegmenten eine Schlüsselumrechnungsroutine benutzen; daher sollten Sie für HDAM-Datenbanken READ PHYSICAL verwenden.

Bei **VSAM-Datenbanken** ist LOGICAL nur möglich für KSDS mit definierten Primär- und Alternativschlüsseln bzw. ESDS mit Alternativschlüsseln.

Wird ein Deskriptorfeld verwendet, das mit Nullwertunterdrückung definiert ist (gilt nur für Adabas), werden Datensätze, bei denen das Deskriptorfeld einen Nullwert enthält, nicht gelesen.

Wird als Deskriptorfeld ein multiples Feld verwendet (gilt nur für Adabas), wird ein einzelner Datensatz mehrmals gelesen, wenn das Feld mehrere Werte enthält.

ASCENDING/DESCENDING/VARIABLE SEQUENCE

Diese Klausel gilt nur für Adabas-, VSAM- und SQL-Datenbanken. Bei einem READ PHYSICAL-Statement gilt sie nur für VSAM-Datenbanken.

Mit dieser Klausel können Sie bestimmen, ob die Datensätze in aufsteigender Reihenfolge oder in absteigender Reihenfolge gelesen werden sollen.

- Standardmäßig werden die Datensätze in aufsteigender Reihenfolge gelesen (was Sie mit dem Schlüsselwort ASCENDING auch ausdrücklich angeben können, aber nicht müssen).
- Wenn die Datensätze in absteigender Reihenfolge gelesen werden sollen, geben Sie das Schlüsselwort DESCENDING an.
- Wenn erst zur Laufzeit bestimmt werden soll, ob die Datensätze in aufsteigender oder absteigender Reihenfolge gelesen werden sollen, geben Sie das Schlüsselwort VARIABLE gefolgt von einer Variablen (*operand5*) an. Der Wert von *operand5* zu Beginn der READ-Verarbeitungsschleife bestimmt dann die Reihenfolge. *Operand5* muß Format/Länge A1 haben und kann den Wert “A” (für “aufsteigend”) oder “D” (für “absteigend”) enthalten.

Anmerkung für Adabas:

Absteigende Lesereihenfolge setzt folgende Adabas-Versionen (oder höher) voraus: Version 3.1 auf UNIX und Windows, Version 3.2 auf OpenVMS bzw. Version 6.1. auf Großrechnern.

Anmerkung für SQL-Datenbanken:

Auf Großrechnern kann die VARIABLE-Option bei SQL-Datenbanken nicht verwendet werden.

Beispiel für DESCENDING:

```
READ EMPLOYEES IN DESCENDING SEQUENCE BY NAME = 'SMITH'
```

Beispiel für VARIABLE:

```
DEFINE DATA LOCAL
1 #DIRECTION (A1) INIT <'A'> /* 'A' = ASCENDING
1 #EMPVIEW VIEW OF EMPLOYEES
  2 NAME
  ...
END-DEFINE
...
IF *PF-KEY = 'PF7'
  THEN MOVE 'D' TO #DIRECTION
END-IF
READ #EMPVIEW IN VARIABLE #DIRECTION SEQUENCE BY NAME = 'SMITH'
...
END-READ
...
```

Weitere Beispiele:

Siehe Programme READSCND und REAVSEQ in Library SYSEXRM.

STARTING FROM / ENDING AT

Mit den Klauseln STARTING FROM und ENDING AT können Sie angeben, ab welchem Wert und bis zu welchem Wert des Deskriptorfeldes gelesen werden soll. Es wird immer ab bzw. bis *einschließlich* der angegebenen Werte gelesen. Wird ein Startwert angegeben, so wird ab diesem Wert gelesen. Wird dieser Wert nicht gefunden, so wird ab dem nächsthöheren Wert gelesen. Ist kein höherer Wert vorhanden, wird die Verarbeitungsschleife nicht durchlaufen.

Mit "EQ" oder "=" wird lediglich ein Startwert angegeben. Zur Angabe eines Endwertes ist eine ENDING AT-Klausel erforderlich.

Ist das Deskriptorfeld ein Adabas-Hyperdeskriptor, darf keine ENDING AT-Klausel verwendet werden.

Bei einem multiplen Feld darf ebenfalls keine ENDING AT-Klausel verwendet werden.

Anmerkung:

Um das Ende des zu lesenden Wertebereichs zu bestimmen, liest Natural intern einen Wert über den ENDING AT-Wert hinaus. Wenn Sie den letzten gelesenen Datensatz weiterverarbeiten, denken Sie bitte daran, daß es sich hierbei nicht um den letzten Datensatz innerhalb des ENDING AT-Bereichs handelt, sondern um den ersten Datensatz jenseits dieses Bereichs (es sei denn, es gibt keinen weiteren Wert nach dem ENDING AT-Wert).

STARTING WITH ISN=*operand4*

Anmerkung:

Diese Klausel gilt nur für Adabas- und VSAM-Datenbanken.

Zugriff auf Adabas

Diese Klausel kann in Verbindung mit einem READ-Statement in physischer oder logischer Reihenfolge (aufsteigend/absteigend) verwendet werden. Der angegebene Wert (*operand4*) steht für eine Adabas ISN (Interne Satznummer) und wird verwendet, um einen bestimmten Datensatz anzugeben, ab dem die READ-Leseschleife gestartet werden soll.

Logische Reihenfolge

Auch wenn das READ-Statement mit einem Gleichheitszeichen “=” versehen ist, gibt es nicht nur diejenigen Datensätze mit genau dem Startwert in dem betreffenden Deskriptorfeld zurück, sondern startet ab dem angegebenen Startwert einen logischen Suchlauf in aufsteigender oder absteigender Reihenfolge. Wenn einige Datensätze im Deskriptorfeld denselben Inhalt haben, werden sie in der Reihenfolge der ISNs sortiert zurückgegeben.

Die Klausel STARTING WITH ISN ist so was wie ein Selektionskriterium der “zweiten Stufe”, das nur gilt, wenn der Startwert mit dem Deskriptorwert für den ersten Datensatz übereinstimmt.

Alle Datensätze mit einem Deskriptorwert, der mit dem Startwert identisch ist, und mit einer ISN, die “kleiner gleich” (“größer gleich” für ein absteigendes READ) der Start-ISN ist, werden von Adabas ignoriert. Der erste in der READ-Schleife zurückgegebene Datensatz ist entweder

- der erste Datensatz mit Deskriptor = Startwert und einer ISN “größer” (“kleiner” für ein absteigendes READ) als die Start-ISN
- oder wenn ein solcher Datensatz nicht vorhanden ist, der erste Datensatz mit einem Deskriptor “größer” (“kleiner” für ein absteigendes READ) als der Startwert.

Physische Reihenfolge

Die Datensätze werden in der Reihenfolge zurückgegeben, in der sie physisch gespeichert sind. Wenn eine STARTING WITH ISN-Klausel angegeben wird, ignoriert Adabas alle Datensätze, bis der Datensatz mit der ISN, die mit der Start-ISN identisch ist, erreicht ist. Der erste zurückgegebene Datensatz ist der nächste auf den Datensatz mit der Start-ISN folgende Datensatz.

Zugriff auf VSAM

Diese Klausel kann nur in physischer Reihenfolge verwendet werden. Der angegebene Wert (*operand4*) steht für eine VSAM RBA (relative Byte-Adresse von ESDS) oder RRN (relative Datensatznummer von RRDS), die als Startwert für die READ-Leseoperation verwendet werden soll.

Beispiele

Diese Klausel kann zum Repositionieren innerhalb einer READ-Schleife, deren Verarbeitung unterbrochen wurde, benutzt werden, um auf einfache Weise den nächsten Datensatz zu bestimmen, mit dem die Verarbeitung fortgesetzt werden soll.

Dies ist besonders hilfreich, wenn der nächste Datensatz sich nicht eindeutig durch einen seiner Deskriptorwerte ermitteln läßt. Es kann auch hilfreich sein in einer verteilten Client/Server-Anwendung, in der das Lesen der Datensätze von einem Server-Programm und die weitere Verarbeitung der Datensätze von einem Client-Programm durchgeführt werden, wobei die Datensätze nicht alle auf einmal, sondern stapelweise verarbeitet werden.

Beispiel:

Siehe Programm REASISND in Library SYSEXRM.

WHERE-Klausel

Mit der WHERE-Klausel können Sie ein zusätzliches Selektionskriterium in Form einer logischen Bedingung (*logical-condition*) angeben. Diese wird ausgewertet, *nachdem* ein Wert gelesen wurde, aber *bevor* eine weitere Verarbeitung auf der Grundlage dieses Wertes (einschließlich AT BREAK-Verarbeitung) erfolgt.

Näheres zu logischen Bedingungen finden Sie im *Natural Referenzhandbuch*.

Ist über ein LIMIT-Statement oder eine Limit-Notation die Anzahl der zu lesenden Datensätze begrenzt, so werden bei einem READ-Statement, das eine WHERE-Klausel enthält, Datensätze, die aufgrund der WHERE-Bedingung nicht weiterverarbeitet werden, bei der Ermittlung des Limits *nicht* mitgezählt.

Systemvariablen

Folgende Natural-Systemvariablen stehen mit dem READ-Statement zur Verfügung:

- ***ISN** — Enthält die Adabas-ISN des gerade verarbeiteten Datensatzes.
Bei VSAM-Datenbanken enthält *ISN entweder die RRN (für RRDS) oder die RBA (für ESDS) des aktuellen Datensatzes.
Bei DL/I- und SQL-Datenbanken oder mit Entire System Server ist *ISN nicht verfügbar.
- ***COUNTER** — Enthält die Anzahl, wie oft die Verarbeitungsschleife durchlaufen wurde.

Format/Länge dieser Systemvariablen ist P10. Format und Länge können nicht geändert werden.

Die Verwendung der Systemvariablen ist im nachstehenden Beispiel illustriert.

Beispiel 1

```

/* EXAMPLE 'REAEX1S': READ (STRUCTURED MODE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
LIMIT 3
/*****
WRITE 'READ IN PHYSICAL SEQUENCE'
READ EMPLOY-VIEW IN PHYSICAL SEQUENCE
  DISPLAY NOTITLE PERSONNEL-ID NAME *ISN *COUNTER
END-READ
/*****
WRITE / 'READ IN ISN SEQUENCE'
READ EMPLOY-VIEW BY ISN
  STARTING FROM 1 ENDING AT 3
  DISPLAY PERSONNEL-ID NAME *ISN *COUNTER
END-READ
/*****
WRITE / 'READ IN NAME SEQUENCE'
READ EMPLOY-VIEW BY NAME
  DISPLAY PERSONNEL-ID NAME *ISN *COUNTER
END-READ
/*****
WRITE / 'READ IN NAME SEQUENCE STARTING FROM 'M''
READ EMPLOY-VIEW BY NAME
  STARTING FROM 'M'
  DISPLAY PERSONNEL-ID NAME *ISN *COUNTER
END-READ
/*****
END

```

Äquivalentes Reporting-Mode-Beispiel: siehe Programm REAEX1R in Library SYSEXRM.

PERSONNEL ID	NAME	ISN	CNT
READ IN PHYSICAL SEQUENCE			
50005600	MORENO	2	1
50005500	BLOND	3	2
50005300	MAIZIERE	4	3
READ IN ISN SEQUENCE			
50005800	ADAM	1	1
50005600	MORENO	2	2
50005500	BLOND	3	3
READ IN NAME SEQUENCE			
60008339	ABELLAN	479	1
30000231	ACHIESON	884	2
50005800	ADAM	1	3
READ IN NAME SEQUENCE STARTING FROM 'M'			
30008125	MACDONALD	929	1
20028700	MACKARNESS	780	2
40000045	MADSEN	509	3

Beispiel 2 (Kombination von READ und FIND)

Das folgende Beispiel verwendet zunächst ein READ-Statement, um Datensätze von der Datei EMPLOYEES in logischer Reihenfolge der Werte des Deskriptorfeldes NAME zu lesen. Dann werden mit einem FIND-Statement Datensätze von der Datei VEHICLES gelesen, wobei die Personalnummer (PERSONNEL-ID) der EMPLOYEES-Datei als Suchkriterium benutzt wird.

Der erzeugte Report zeigt den Namen und die Personalnummer aller von der EMPLOYEES-Datei gelesenen Personen sowie die Automobilmarken (MAKE) der Autos (von der VEHICLES-Datei), die im Besitz dieser Personen sind. Besitzt eine Person mehrere Autos, werden entsprechend mehrere Zeilen pro Person ausgegeben.

```

/* EXAMPLE 'REAEX2': READ AND FIND
DEFINE DATA
  LOCAL
  1 EMPLOY-VIEW VIEW OF EMPLOYEES
    2 PERSONNEL-ID
    2 FIRST-NAME
    2 NAME
    2 CITY
  1 VEH-VIEW VIEW OF VEHICLES
    2 PERSONNEL-ID
    2 MAKE
END-DEFINE
LIMIT 10
RD. READ EMPLOY-VIEW BY NAME STARTING FROM 'JONES'
  SUSPEND IDENTICAL SUPPRESS
FD.   FIND VEH-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (RD.)
      IF NO RECORDS FOUND
        ENTER
      END-NOREC
      DISPLAY NOTITLE (ES=OFF IS=ON ZP=ON AL=15)
        PERSONNEL-ID (RD.) FIRST-NAME (RD.)
        MAKE (FD.) (IS=OFF)
      END-FIND
    END-READ
END

```

PERSONNEL ID	FIRST-NAME	MAKE
20007500	VIRGINIA	CHRYSLER
20008400	MARSHA	CHRYSLER
		CHRYSLER
20021100	ROBERT	GENERAL MOTORS
20000800	LILLY	FORD
		MG
20001100	EDWARD	GENERAL MOTORS
20002000	MARTHA	GENERAL MOTORS
20003400	LAUREL	GENERAL MOTORS
30034045	KEVIN	DATSUN
30034233	GREGORY	FORD
11400319	MANFRED	

READ WORK FILE

Structured-Mode-Syntax

READ WORK [FILE] *work-file-number* [ONCE]

{ **RECORD** *operand1*
 { [AND] [SELECT] { [**OFFSET** *n*
 FILLER *nX*] ... *operand2* } ... } }

[GIVING LENGTH *operand3*]

{ **AT** [END] [OF] [FILE]
 statement...
 END-ENDFILE }

statement...

END-WORK

Reporting-Mode-Syntax

READ WORK [FILE] *work-file-number* [ONCE]

{ **RECORD** { *operand1* [FILLER *nX*] } ...
 { [AND] [SELECT] { [**OFFSET** *n*
 FILLER *nX*] ... *operand2* } ... } }

[GIVING LENGTH *operand3*]

[**AT** [END] [OF] [FILE] { *statement*
 DO *statement...* **DOEND** }]

statement...

[LOOP]

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S A G	A N P I F B D T L C G	ja	ja
Operand2	S A G	A N P I F B D T L C	ja	ja
Operand3	S	I	ja	ja

Verwandte Statements

DEFINE WORK FILE, CLOSE WORK FILE.

Funktion

Das Statement READ WORK FILE dient dazu, Daten von einer physisch-sequentiellen Nicht-Adabas-Arbeitsdatei zu lesen. Die Daten werden sequentiell von der Arbeitsdatei gelesen. Wie sie gelesen werden, ist unabhängig davon, wie Sie auf die Arbeitsdatei geschrieben wurden.

Auf Großrechnern kann dieses Statement nur in einem Programm verwendet werden, das unter Com-plete, CMS, TSO, TIAM oder im Batch-Betrieb ausgeführt wird. Die entsprechende JCL muß mit der Ausführungs-JCL bereitgestellt werden, wenn eine Arbeitsdatei gelesen werden soll. Näheres hierzu siehe *Natural Operations Documentation for Mainframes*.

Das READ WORK FILE-Statement führt eine Verarbeitungsschleife aus, um alle Datensätze der Arbeitsdatei zu lesen. Innerhalb einer READ WORK FILE-Schleife können automatische Gruppenwechsel-Verarbeitungen durchgeführt werden.

Anmerkung:

Wenn bei Ausführung eines READ WORK FILE-Statements eine "End-of-File"-Bedingung auftritt, schließt Natural die Arbeitsdatei automatisch.

Anmerkung für Entire Connection:

Beim Lesen von Entire Connection-Arbeitsdateien darf innerhalb der READ WORK FILE-Verarbeitungsschleife kein I/O-Statement stehen.

work-file-number

Die für Natural definierte Nummer der Arbeitsdatei, die gelesen werden soll.

ONCE-Option

Die ONCE-Option bewirkt, daß nur *ein* Datensatz gelesen und keine Verarbeitungsschleife initiiert wird (das schleifenbeendende Schlüsselwort END–WORK bzw. LOOP darf daher nicht zusammen mit ONCE verwendet werden). Wenn Sie ONCE verwenden, sollten Sie auch eine AT END OF FILE-Klausel verwenden.

Wird ein READ WORK FILE-Statement mit ONCE-Option von einer benutzerinitiierten Verarbeitungsschleife gesteuert, kann es sein, daß auf der Arbeitsdatei eine “End-of-File”-Bedingung auftritt, bevor die Schleife beendet wird. Alle von der Arbeitsdatei gelesenen Felder enthalten dann immer noch die Werte des zuletzt gelesenen Datensatzes. Die Arbeitsdatei wird dann zum ersten Datensatz zurückpositioniert, welcher dann bei der nächsten Ausführung des READ WORK FILE ONCE-Statements gelesen wird.

Variabler Indexbereich

Wenn Sie ein Array von einer Arbeitsdatei lesen, können Sie für das Array einen variablen Indexbereich angeben. Zum Beispiel:

```
READ WORK FILE work-file-number #ARRAY (I:J)
```

RECORD-Option

Wird RECORD angegeben, so werden alle Felder jedes gelesenen Datensatzes zur Verarbeitung zur Verfügung gestellt. Entsprechend dem Aufbau des Datensatzes muß eine Operandenliste (*operand1*) angegeben werden. Mit FILLER *nX* geben Sie an, daß beim Eingabedatensatz *n* Bytes übersprungen werden sollen. Der in der RECORD-Klausel definierte Datensatz muß in einem zusammenhängenden Speicherbereich gespeichert sein. Im *Structured Mode* ist FILLER nicht erlaubt.

Im *Structured Mode* — oder wenn *operand1* in einem DEFINE DATA-Statement definiert ist — darf *operand1* nur einmal angegeben werden. In beiden Fällen darf FILLER nicht verwendet werden.

Natural überprüft die Daten des Datensatzes nicht. Demzufolge ist diese Option die schnellste Art, Datensätze von einer sequentiellen Datei zu verarbeiten. Allerdings müssen Sie selbst darauf achten, daß der Aufbau des Datensatzes korrekt beschrieben ist, da es sonst durch nicht-numerische Daten in numerischen Feldern zu einem ungewollten Programmabbruch kommen kann. Bevor der Datensatz gelesen wird, füllt Natural die Operandenliste (*operand1*) mit Leerzeichen; demzufolge wird bei einer “End-of-File”-Bedingung ein leerer Bereich zurückgegeben. Kurze Datensätze werden mit Leerzeichen aufgefüllt.

Anmerkung für Entire Connection:

Beim Lesen von Entire Connection-Arbeitsdateien kann die RECORD-Option nicht verwendet werden.

SELECT-Option (Standard)

Wird die SELECT-Option verwendet, so werden nur die in der Operandenliste (*operand2*) angegebenen Felder zur Verfügung gestellt. Die Position der Felder im Eingabedatensatz kann durch eine OFFSET- und/oder FILLER-Angabe spezifiziert werden. OFFSET 0 spezifiziert das erste Byte des Datensatzes. FILLER *nX* bedeutet, daß *n* Bytes des Eingabedatensatzes übersprungen werden.

Natural ordnet den einzelnen Feldern die ausgewählten Werte zu und überprüft, daß die vom Datensatz ausgewählten numerischen Felder entsprechend ihrer Definition auch wirklich gültige numerische Werte enthalten. Aufgrund dieser Prüfung dauert die Verarbeitung einer sequentiellen Datei mit der SELECT-Option etwas länger als mit der RECORD-Option.

Wenn ein Datensatz nicht alle in der SELECT-Option angegebenen Felder füllt, gilt folgendes:

- Ein Feld, das nur teilweise gefüllt wurde, wird mit Leerzeichen bzw. Nullen aufgefüllt.
- Ein Feld, das gar nicht gefüllt wurde, behält denselben Inhalt wie zuvor.

Anmerkung für Entire Connection:

Beim Lesen von Entire Connection-Arbeitsdateien wird die OFFSET-Option ignoriert.

Feldlängen

Die Länge der Felder in der Operandenliste wird wie folgt bestimmt:

- Bei Feldern der Formate A, B, I und F entspricht die Anzahl der Bytes im Eingabedatensatz der internen Längendefinition.
- Bei Feldern des Formats N ergibt sich die Anzahl der Bytes im Eingabedatensatz aus der Summe der internen Stellen vor und nach dem Komma (Dezimalpunkt). Komma und Vorzeichen belegen im Eingabedatensatz kein Byte.
- Bei Feldern der Formate P, D und T ergibt sich die Anzahl der Bytes im Eingabedatensatz aus der Summe der Stellen vor und nach dem Komma (Dezimalpunkt) plus einer Stelle für das Vorzeichen, geteilt durch 2 und aufgerundet.
- Bei Feldern des Formats L wird 1 Byte benutzt.
- Bei Feldern des Formats C werden 2 Bytes benutzt.

Beispiele für Feldlängen:

Felddefinition	Eingabedatensatz
#FIELD1 (A10)	10 Bytes
#FIELD2 (B15)	15 Bytes
#FIELD3 (N1.3)	4 Bytes
#FIELD4 (N0.7)	7 Bytes
#FIELD5 (P1.2)	2 Bytes
#FIELD6 (P6.0)	4 Bytes

Vgl. Abschnitt **Definition von Format und Länge** im *Natural Referenzhandbuch*.

GIVING LENGTH *operand3*

Mit der GIVING LENGTH-Klausel können Sie die tatsächliche Länge des gelesenen Datensatzes in Erfahrung bringen. Die Länge (Anzahl der Bytes) erhalten Sie in *operand3*. *Operand3* muß mit Format/Länge I4 definiert sein.

AT END OF FILE

Die AT END OF FILE-Klausel kann nur zusammen mit der ONCE-Option verwendet werden. Wenn Sie die ONCE-Option verwenden, dann sollten Sie auch eine AT END OF FILE-Klausel angeben, um festzulegen, was beim Auftreten einer “End-of-File”-Bedingung geschehen soll.

Wenn Sie die ONCE-Option nicht verwenden, wird eine “End-of-File”-Bedingung wie das normale Ende einer Verarbeitungsschleife behandelt.

Verarbeitung großer und dynamischer Variablen

Die Option RECORD ist nicht zulässig, wenn dynamische Variablen verwendet werden.

Die folgenden Arbeitsdatei-Typen ASCII, ASCII-COMPRESSED, ENTIRECONNECTION; SAG (binary) und TRANSFER können dynamische Variablen nicht verarbeiten, was zu einem Fehler führt. Große Variablen stellen kein Problem dar, außer wenn die maximale Feld/Datensatz-Länge überschritten wird (Feldlänge 255 für ENTIRECONNECTION und TRANSFER, Datensatz-Länge 32767 für die anderen).

Das Lesen einer dynamischen Variable von einer “PORTABLE”-Arbeitsdatei führt zu einer Änderung auf die gespeicherte Länge. Das Lesen einer dynamischen Variable von einer “UNFORMATTED”-Arbeitsdatei bewirkt, daß der komplette Rest der Datei in die Variable (von der aktuellen Position) gestellt wird. Wenn die Datei mehr als 1 GB umfaßt, dann werden zumindestens 1 GB in die Variable gestellt.

Beispiel

```

/* EXAMPLE 'RWFE1': READ WORK FILE
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
1 #RECORD
  2 #PERS-ID (A8)
  2 #NAME (A20)
END-DEFINE
*****
FIND EMPLOY-VIEW WITH CITY = 'STUTTGART'
WRITE WORK FILE 1
  PERSONNEL-ID
  NAME
END-FIND
*****
/* ...
*****
READ WORK FILE 1
  RECORD
  #RECORD
  DISPLAY NOTITLE #PERS-ID #NAME
END-WORK
*****
END

```

PAGE 1

87-03-27 15:46:58

```

11100328 BERGHAUS
11100329 BARTHEL
11300313 AECKERLE
11300316 KANTE
11500304 KLUGE
11500308 DIETRICH
11500318 GASSNER
11500343 ROEHM
11600303 BERGER
11600320 BLAETTEL
11500336 JASPER
11100330 BUSH
11500328 EGGERT

```

REDEFINE

Anmerkung:

Dieses Statement gilt nur im Reporting Mode. Um ein Feld im Structured Mode zu redefinieren, verwenden Sie das DEFINE DATA-Statement.

$$\text{REDEFINE } \left\{ \text{operand1 } \left(\left\{ \begin{matrix} nX \\ \text{operand2} \end{matrix} \right\} \dots \right) \right\} \dots$$

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S A G	A N P I F B D T L C	ja	nein
Operand2	S A G	A N P I F B D T L C	ja	ja

Funktion

Das Statement REDEFINE dient dazu, ein Feld zu redefinieren. Das Ergebnis der Neudefinition können eine oder mehrere Benutzervariablen sein.

Mit einem REDEFINE-Statement können Sie gleichzeitig mehrere Felder redefinieren.

Methode der Redefinition

Die Byte-Positionen von *operand1* werden unabhängig vom Format von links nach rechts redefiniert. Das Format von *operand2* muß nicht mit dem von *operand1* identisch sein. Die Byte-Positionen des neudefinierten Feldes müssen zu den im Feld enthaltenen Daten passen; wird beispielsweise ein alphanumerisches Feld als numerisch redefiniert, und es enthält keine numerischen Daten, so kann die Verwendung des Feldes einen Programmabbruch zur Folge haben.

Weitere Redefinition

Ein mit einem REDEFINE-Statement neudefiniertes Feld kann mit einem weiteren REDEFINE-Statement nochmals redefiniert werden.

Füllbyte-Notation

Mit der Notation nX können Sie in der redefinierten Variable n Füllbytes definieren. Nachgestellte Füllbytes müssen nicht unbedingt angegeben werden.

Beispiel 1

Die Benutzervariable #A (Format/Länge A10) enthält den Wert "123ABCDEFGG".

```
REDEFINE #A (#A1(N3) #A2(A7))
```

#A1 erhält den Wert "123", #A2 den Wert "ABCDEFGG".

Beispiel 2

Die Benutzervariable #B (Format/Länge A10) enthält den hexadezimalen Wert "12345CC1C2C3C4C5C6C7".

```
REDEFINE #B (#B1(P4) #B2(A7))
```

#B1 erhält den hexadezimalen Wert "12345C",
#B2 den hexadezimalen Wert "C1C2C3C4C5C6C7".

```
REDEFINE #B (#BB1(B2)8X) oder REDEFINE #B(#BB1(B2))
```

#BB1 erhält den hexadezimalen Wert "1234".

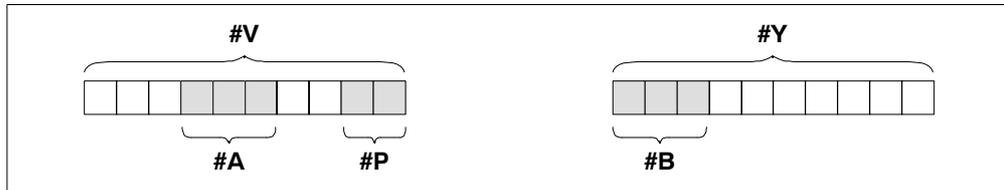
Anmerkung:

Beim Format P (gepackt numerisch) muß die Anzahl der benötigten Dezimalstellen angegeben werden. Die Anzahl der Bytes, die eine gepackte Zahl benötigt, läßt sich wie folgt berechnen:

Anzahl der Bytes = (Anzahl der Dezimalstellen + 1) / 2, auf ganze Bytes aufgerundet.

Beispiel 3

```
COMPUTE #V (N8.2) = #Y (N10) = ...
REDEFINE #V (3X #A(N3) 2X #P (N2)) #Y (#B(N3) 7X)
```



Beispiel 4

In diesem Beispiel wird die Systemvariable *DATN, die die Form “YYYYMMDD” hat, redefiniert und das Ergebnis in der Reihenfolge Tag/Monat/Jahr (DAY/MONTH/YEAR) in drei getrennte Felder geschrieben:

```
MOVE *DATN TO #DATINT (N8)
REDEFINE #DATINT (#YEAR (N4) #MONTH (N2) #DAY (N2))
DISPLAY NOTITLE #DATINT #DAY #MONTH #YEAR
END
```

#DATINT	#DAY	#MONTH	#YEAR
19950108	8	1	1995

REDUCE

Anmerkung:

Dieses Statement steht auf Großrechnern nicht zur Verfügung.

REDUCE [SIZE OF] DYNAMIC [VARIABLE] operand1 TO operand2

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S	A B	nein	nein
Operand2	C S	I	nein	nein

Verwandte Statements

EXPAND.

Funktion

Das Statement REDUCE DYNAMIC VARIABLE dient dazu, die zugewiesene Länge einer dynamischen Variable (*operand1*) auf die angegebene Länge (*operand2*) zu verringern. Liegt der zugewiesene Speicherplatz der dynamischen Variable oberhalb der angegebenen Länge, wird er sofort freigegeben, d.h. dann, wenn das Statement ausgeführt wird.

Wenn die aktuell verwendete Länge (*LENGTH) der dynamischen Variable die angegebene Länge übersteigt, wird *LENGTH auf die angegebene Länge gesetzt, und der Inhalt der Variable wird abgeschnitten (aber nicht geändert). Liegt die angegebene Länge über der aktuell zugewiesenen Länge der dynamischen Variable, wird das Statement ignoriert.

operand1

Operand1 ist die dynamische Variable, für die die zugewiesene Länge verringert werden soll.

operand2

Operand2 dient dazu, die verringerte Länge anzugeben. Der angegebene Wert darf kein negativer numerischer Wert sein.

Anmerkung:

Der "MODIFIED"-Status einer in dem betreffenden INPUT-Statement verwendeten Kontrollvariablen wird bei der Ausführung von einem REINPUT-Statement (ohne FULL-Option) nicht zurückgesetzt.

REINPUT FULL

Wenn Sie die Option FULL in einem REINPUT-Statement angeben, wird das entsprechende INPUT-Statement vollständig neu ausgeführt:

- Bei einem normalen REINPUT-Statement (ohne FULL-Option) werden Inhalte von Variablen, die zwischen INPUT- und REINPUT-Statement geändert wurden, nicht angezeigt; d.h. alle Variablen auf dem Schirm zeigen den Inhalt, den sie hatten, als das INPUT-Statement ursprünglich ausgeführt wurde.
- Bei einem REINPUT FULL-Statement werden alle nach der ersten Ausführung des INPUT-Statements gemachten Änderungen sichtbar, wenn das INPUT-Statement erneut ausgeführt wird; d.h. alle Variablen auf dem Schirm haben den Inhalt, den sie zum Zeitpunkt der Ausführung des REINPUT-Statements hatten.

Anmerkung:

Der Inhalt reiner Eingabefelder (AD=A) wird durch REINPUT FULL wieder gelöscht.

statement-parameters

Mit einem REINPUT-Statement gesetzte Parameter gelten für alle Felder, die im Statement angegeben sind.

Auf Feldebene gesetzte Parameter (siehe MARK-Option) haben für das betreffende Feld Gültigkeit vor auf Statement-Ebene gesetzten.

Wird AD=P auf Statement-Ebene gesetzt, so sind alle Felder geschützt, außer den in der MARK-Option angegebenen.

Informationen zu den einzelnen Parametern finden Sie im Kapitel **Session-Parameter** des *Natural Referenzhandbuchs*.

USING HELP

USING HELP bewirkt, daß die für die INPUT-Map definierte Helproutine aufgerufen wird.

USING HELP in Verbindung mit der MARK-Option (siehe unten) bewirkt, daß die für das erste in der MARK-Option angegebene Feld definierte Helproutine aufgerufen wird. Ist für das Feld keine Helproutine definiert, wird die Helproutine für die Map aufgerufen.

Beispiel:

```
REINPUT USING HELP MARK 3
```

In diesem Beispiel wird die für das dritte Feld der INPUT-Map definierte Helproutine aufgerufen.

WITH TEXT-option

$$[\text{WITH}] [\text{TEXT}] \left\{ \begin{array}{l} *operand1 \\ operand2 \end{array} \right\} [(attributes)] [,operand3] \dots_7$$

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	N P I B	ja	nein
Operand2	C S	A	ja	nein
Operand3	C S	A N P I F B D T L	ja	nein

Mit dieser Option können Sie einen Text angeben, der in der Meldungszeile angezeigt werden soll. Der Text ist in der Regel eine Meldung, die dem Benutzer sagt, wie er den Fehler korrigieren kann.

Meldungstext aus der Natural-Fehlermeldungsdatei (*operand1)

Als *operand1* geben Sie eine Natural-Fehlernummer an. Natural liest dann die entsprechende Fehlermeldung von der Natural-Fehlermeldungsdatei.

In der Fehlermeldungsdatei sind die Fehlermeldungen nach Libraries sortiert gespeichert. Pro Library können bis zu 9999 Meldungen gespeichert werden.

Weitere Informationen zu Fehlermeldungen finden Sie in Ihrer *Natural SYSERR Utility Documentation* unter **Generating Message Files**.

Meldungstext (*operand2*) und Attribute (*attributes*)

Als *operand2* geben Sie den Text an, der in der Meldungszeile ausgegeben werden soll.

Als *attributes* können Sie *operand1* oder *operand2* bestimmte Anzeige- und Farbattribut zuordnen. Sie können folgende *attributes* angeben:



Mit AD= geben Sie ein Anzeigeattribut an; mit CD= geben Sie ein Farbattribut an (vgl. Session-Parameter AD bzw. CD im *Natural Referenzhandbuch*).

Dynamische Meldungstext-Komponente (*operand3*)

Operand3 kann in Form einer numerischen Konstanten oder Textkonstanten oder als Name einer Variablen angegeben werden.

Der angegebene Wert dient dazu, einen Teil der Meldung dynamisch zu generieren.

Innerhalb der Fehlermeldung dient die Notation “:n:” zur Referenzierung von *operand3*, wobei *n* die Ausprägung (1 – 7) von *operand3* darstellt.

Beispiel:

```
...  
MOVE 'MESSAGE-1' TO #FIELD  
...  
REINPUT 'THE ERROR IS :1:',#FIELD  
...
```

Dies würde die Ausgabe folgender Meldung bewirken:

THE ERROR IS MESSAGE-1

Anmerkung:

Werden mehrere Operanden3 angegeben, müssen diese mit einem Komma voneinander getrennt werden. Falls das Komma als Dezimalzeichen verwendet wird (wie mit dem Session-Parameter DC definiert) und es sich bei Operand3 um numerische Konstanten handelt, setzen Sie Leerzeichen vor und nach dem Komma, damit es nicht als Dezimalkomma mißinterpretiert wird.

Alternativ können mehrere Operanden3 auch mit dem Input-Delimiterzeichen (wie mit dem Session-Parameter ID definiert) voneinander getrennt werden; dies geht jedoch nicht im Falle von ID=/ (Schrägstrich).

Nicht signifikante Nullen oder Leerzeichen werden aus dem Feldwert entfernt, bevor er in einer Meldung angezeigt wird.

MARK-option

$\text{MARK} \left[\text{POSITION } \textit{operand4} \text{ [IN]} \right] \left[\text{FIELD} \left\{ \left\{ \begin{array}{l} \textit{operand5} \\ *fieldname \end{array} \right\} \left[(\textit{attributes}) \right] \right\} \dots \right]$
--

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand4	C S	N P I	ja	nein
Operand5	C S A	N P I	ja	nein

Mit der MARK-Option können Sie ein bestimmtes Feld markieren, d.h. bei der Ausführung des REINPUT-Statements wird der Cursor in dieses Feld plaziert. Sie können auch eine bestimmte Stelle innerhalb eines Feldes markieren. Außerdem können Sie Felder gegen Eingabe schützen sowie ihre Anzeige- und Farbattribute ändern.

Das zu markierende Feld (*operand5*)

Jedes mit einem INPUT-Statement angegebene Eingabefeld (AD=A oder AD=M) wird durchnummeriert (beginnend mit 1). Sie können als *operand5* die Nummer des Feldes angeben, in das der Cursor plaziert werden soll.

Die Notation **fieldname* wird verwendet, um den Cursor auf ein Feld zu positionieren, und zwar mittels des (im INPUT-Statement verwendeten) Namens des Feldes.

Ist das betreffende INPUT-Feld ein Array, kann zur Markierung einer oder mehrerer Ausprägungen des Arrays ein eindeutiger Index oder ein Indexbereich angegeben werden.

```
INPUT #ARRAY (A1/1:5)
...
REINPUT (AD=P) 'TEXT' MARK *#ARRAY (2:3)
```

Ist *operand5* ebenfalls ein Array, so werden die Werte von *operand5* als Feldnummern für das INPUT-Array benutzt.

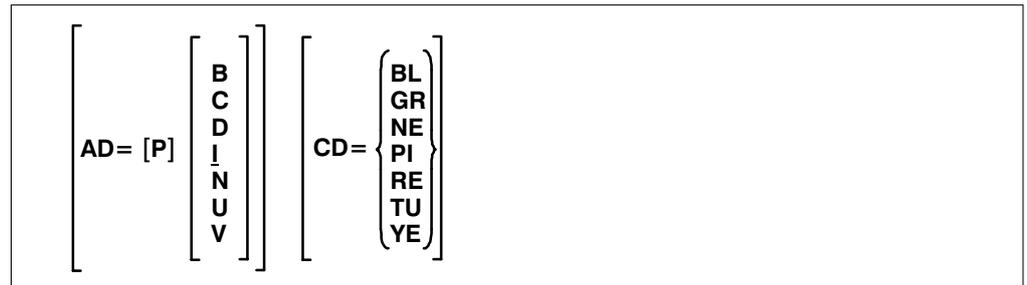
```
RESET #X(N2/1:2)
INPUT #ARRAY ...
...
REINPUT (AD=P) 'TEXT' MARK #X (1:2)
```

MARK POSITION

Mit MARK POSITION können Sie den Cursor an eine bestimmte Stelle — die Sie mit *operand4* angeben — innerhalb eines Feldes plazieren.

Operand4 darf keine Dezimalstellen enthalten.

attributes



Mit dem Attribut AD=P können Sie ein Eingabefeld (AD=A oder AD=M) gegen Eingaben schützen.

Anmerkung:

Reine Ausgabefelder (AD=O) können nicht durch ein entsprechendes Attribut zu Eingabefeldern gemacht werden.

Wird AD=P auf Statement-Ebene gesetzt, so sind alle Felder geschützt außer den in der MARK-Option angegebenen.

Außerdem können Sie Anzeige- und Farbattribute von Feldern ändern. Informationen zu diesen Attributen finden Sie unter Session-Parameter AD bzw. CD im *Natural Referenzhandbuch*.

ALARM-option

[AND] [SOUND] ALARM

Diese Option bewirkt, daß der Warnton des Terminals ausgelöst wird, wenn das REINPUT-Statement ausgeführt wird. Voraussetzung ist, daß die verwendete Terminal-Hardware dies ermöglicht.

Beispiel 1

```

/* EXAMPLE 'REIEX1': REINPUT
/*****
/* IF FUNCTION = A AND PARM = X
/*   ROUTINE-A IS TO BE EXECUTED.
/* IF FUNCTION = B AND PARM = X
/*   ROUTINE-B IS TO BE EXECUTED.
/* IF FUNCTION = C THRU D
/*   ROUTINE-CD IS TO BE EXECUTED.
/* FOR ALL OTHER CASES,
/*   REINPUT STATEMENT IS TO BE EXECUTED.
/*****
DEFINE DATA LOCAL
1 #FUNCTION (A1)
1 #PARM (A1)
END-DEFINE
/*****
INPUT #FUNCTION #PARM
/*****
DECIDE FOR FIRST CONDITION
    WHEN #FUNCTION = 'A' AND #PARM = 'X'
        PERFORM ROUTINE-A
    WHEN #FUNCTION = 'B' AND #PARM = 'X'
        PERFORM ROUTINE-B
    WHEN #FUNCTION = 'C' THRU 'D'
        PERFORM ROUTINE-CD
    WHEN NONE
        REINPUT 'PLEASE ENTER A VALID FUNCTION'
        MARK *#FUNCTION
END-DECIDE
/*****
END

```

```
#FUNCTION a #PARM y
```

```
PLEASE ENTER A VALID FUNCTION
#FUNCTION a #PARM y
```

Beispiel 2

```
/* EXAMPLE 'REIEX2': REINPUT WITH ATTRIBUTE ASSIGNMENT
/*****
DEFINE DATA LOCAL
1 #A (A20)
1 #B (N7.2)
1 #C (A5)
1 #D (N3)
END-DEFINE
/*****
INPUT (AD=A)
  #A #B #C #D
/*****
IF #A = ' ' OR #B = 0
  REINPUT (AD=P) 'RETYPE VALUES'
    MARK *#A (AD=I CD=RE)
    *#B (AD=U CD=PI)
END-IF
/*****
END
```

Beispiel 3

```

/* EXAMPLE 'REIEX3': REINPUT FULL WITH POSITION
/*****
DEFINE DATA LOCAL
1 #A (A20)
1 #B (N7.2)
1 #C (A5)
1 #D (N3)
END-DEFINE
/*****
INPUT (AD=M)
  #A #B #C #D
IF #A = ' '
  COMPUTE #B = #B + #D
  RESET #D
END-IF
/*****
IF #A = SCAN 'TEST' OR = ' '
  REINPUT FULL 'RETYPE VALUES' MARK POSITION 5 IN *#A
END-IF
/*****
END

```

#A	#B	0.00	#C	#D	0
----	----	------	----	----	---

RETYPE VALUES

REJECT

Siehe Statement ACCEPT/REJECT.

RELEASE

RELEASE { <table style="display: inline-table; vertical-align: middle;"> <tr> <td style="text-align: center;">STACK</td> </tr> <tr> <td style="text-align: center;">SETS [<i>set-name...</i>]</td> </tr> <tr> <td style="text-align: center;">VARIABLES</td> </tr> </table>	STACK	SETS [<i>set-name...</i>]	VARIABLES
STACK			
SETS [<i>set-name...</i>]			
VARIABLES			

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Set-name	C S	A	nein	nein

Verwandte Statements

STACK, FIND mit RETAIN-Option, DEFINE DATA GLOBAL.

Funktion

Das Statement RELEASE dient dazu:

- den kompletten Inhalt des Natural-Stacks zu löschen
- Sätze von ISNs freizugeben, die über ein FIND-Statement mit RETAIN-Klausel zurückgehalten wurden (gilt nur für Adabas-Datenbanken)
- globale und applikationsunabhängige Variablen auf Ausgangswerte zurückzusetzen.

RELEASE STACK

Mit RELEASE STACK löschen Sie alle gerade im Natural-Stack gelagerten Kommandos und Daten.

RELEASE SET

RELEASE SET gilt nur für Adabas-Datenbanken.

Mit RELEASE SET *set-name* geben Sie eine bestimmte ISN frei.

```
RELEASE SET 'CITY-SET'  
  
MOVE 'CITY-SET' TO #SET(A32)  
RELEASE SET #SET
```

Wenn Sie nur RELEASE SETS — ohne Angabe eines *set-name* — angeben, werden alle ISNs freigegeben, die mit einem FIND-Statement, das eine RETAIN-Klausel enthält, gehalten wurden.

RELEASE VARIABLES

Mit RELEASE VARIABLES werden alle in der aktuellen Global Data Area definierten Variablen auf ihre Ausgangswerte zurückgesetzt. Gleichzeitig werden alle AIVs (applikationsunabhängigen Variablen) gelöscht, d.h. sie stehen dann nicht mehr zur Verfügung.

Die Variablen werden entweder bei Abschluß der Ausführung des Level-1-Programms zurückgesetzt/gelöscht oder wenn das Programm über ein FETCH- oder RUN-Statement ein anderes Programm aufruft.

Beispiel

```

* EXAMPLE 'RELEX1': FIND (RETAIN CLAUSE) AND RELEASE
*****
DEFINE DATA LOCAL
  1 EMPLOY-VIEW VIEW OF EMPLOYEES
    2 CITY
    2 BIRTH
    2 NAME
  1 #BIRTH (D)
END-DEFINE
*
MOVE EDITED '19400101' TO #BIRTH (EM=YYYYMMDD)
*
FIND NUMBER EMPLOY-VIEW WITH BIRTH GT #BIRTH
      RETAIN AS 'AGESET1'

IF *NUMBER = 0
  STOP
END-IF
*
FIND EMPLOY-VIEW WITH 'AGESET1' AND CITY = 'NEW YORK'
  DISPLAY NOTITLE NAME CITY BIRTH (EM=YYYY-MM-DD)
END-FIND
*
RELEASE SET 'AGESET1'
END

```

NAME	CITY	DATE OF BIRTH
RUBIN	NEW YORK	1945-10-27
WALLACE	NEW YORK	1945-08-04

REPEAT

Syntax 1

```
REPEAT
  statement... [ { UNTIL } logical-condition ]
               [ { WHILE } ]
END-REPEAT    (nur im Structured Mode)
LOOP          (nur im Reporting Mode)
```

Syntax 2

```
REPEAT
  [ { UNTIL } logical-condition ] statement...
  [ { WHILE } ]
END-REPEAT    (nur im Structured Mode)
LOOP          (nur im Reporting Mode)
```

Verwandtes Statement

FOR.

Funktion

Mit dem Statement REPEAT können Sie eine Verarbeitungsschleife initiieren.

Wenn Sie keine logische Bedingung angeben, müssen Sie die Schleife mit einem ESCAPE-, STOP- oder TERMINATE-Statement beenden. Wie oft die Schleife durchlaufen wird, hängt von der von Ihnen angegebenen logischen Bedingung (*logical-condition*) ab.

Wenn Sie Syntax 1 verwenden, werden die Statements mindestens einmal ausgeführt.

Wenn Sie Syntax 2 verwenden, kann es sein, daß die Statements möglicherweise (je nach logischer Bedingung) gar nicht ausgeführt werden.

Wann die logische Bedingung auf ihre Gültigkeit untersucht wird, hängt davon ab, ob Sie sie vor oder nach den im Rahmen der Schleife auszuführenden Statements plazieren.

Weitere Informationen zu logischen Bedingungen finden Sie im Abschnitt **Logische Bedingungen** im *Natural Referenzhandbuch*.

UNTIL

Die Schleife wird so oft ausgeführt, bis die logische Bedingung erfüllt ist.

WHILE

Die Schleife wird solange ausgeführt, wie die logische Bedingung erfüllt ist.

Beispiel 1

```

/* EXAMPLE 'RPTEX1S': REPEAT (STRUCTURED MODE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
1 #PERS-NR (A8)
END-DEFINE
/*****
REPEAT
  INPUT 'ENTER A PERSONNEL NUMBER:' #PERS-NR
  IF #PERS-NR = ' '
    ESCAPE BOTTOM
  END-IF
  FIND EMPLOY-VIEW WITH PERSONNEL-ID = #PERS-NR
  IF NO RECORD FOUND
    REINPUT 'NO RECORD FOUND'
  END-NOREC
  DISPLAY NOTITLE NAME
  END-FIND
END-REPEAT
/*****
END

```

```
ENTER A PERSONNEL NUMBER: 11500304
```

```
NAME
```

```
KLUGE
```

Äquivalentes Reporting-Mode-Beispiel: siehe Programm RPTEX1R in Library SYSEXRM.

Beispiel 2

```

/* EXAMPLE 'RPTEX2S': REPEAT (WHILE AND UNTIL OPTIONS)
/*****
DEFINE DATA LOCAL
1 #X (I1) INIT <0>
1 #Y (I1) INIT <0>
END-DEFINE
/*****
REPEAT WHILE #X <= 5
  ADD 1 TO #X
  WRITE NOTITLE '=' #X
END-REPEAT
/*****
SKIP 1
REPEAT
  ADD 1 TO #Y
  WRITE '=' #Y
  UNTIL #Y = 6
END-REPEAT
/*****
END

```

```

#X: 1
#X: 2
#X: 3
#X: 4
#X: 5
#X: 6

#Y: 1
#Y: 2
#Y: 3
#Y: 4
#Y: 5
#Y: 6

```

Äquivalentes Reporting-Mode-Beispiel: siehe Programm RPTEX2R in Library SYSEXRM.

REQUEST DOCUMENT

Anmerkung:

Dieses Statement ist nur unter UNIX, OpenVMS und Windows verfügbar.

REQUEST DOCUMENT FROM *operand1*

```

{
  WITH
  [USER operand2 ]
  [PASSWORD operand2 ]
  [HEADER [[NAME] operand4 [VALUE] operand5 ]... ]
  [ DATA { [[NAME] operand7 [VALUE] operand8 ]... } ]
}

{
  RETURN
  [HEADER [ALL operand9 ]
  [[NAME] operand10 [VALUE] operand11 ]... ]
  [PAGE operand12 ]
}

```

RESPONSE *operand13*

[GIVING *operand14*]

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	nein	nein
Operand2	C S	A	nein	nein
Operand3	C S	A	nein	nein
Operand4	C S	A	nein	nein
Operand5	C S	A N P I F D T L	nein	nein
Operand6	C S	A N P I F B D T L	nein	nein
Operand7	C S	A	nein	nein
Operand8	C S	A N P I F D T L	nein	nein
Operand9	S	A	nein	ja
Operand10	C S	A	nein	ja
Operand11	S	A N P I F B D T L	nein	ja
Operand12	S	A B	nein	ja
Operand13	S	I	nein	ja
Operand14	S	I	nein	nein

Funktion

Das Statement REQUEST DOCUMENT gibt Ihnen die Möglichkeit, auf ein externes System zuzugreifen.

Beschränkungen für Cookies

Unter dem HTTP-Protokoll verwendet ein Server Cookies, um Status-Informationen über die Client-Workstation zu verwalten.

Bei der Implementierung von REQUEST DOCUMENT unter Windows werden die Internet-Optionseinstellungen verwendet. Dies bedeutet, dass in Abhängigkeit von den Sicherheitseinstellungen Cookies verarbeitet werden.

Wenn “Disabled” (ausgeschaltet) eingestellt ist, werden keine Cookies übermittelt, auch wenn ein COOKIE-Header (*operand 4/5*) übermittelt wird.

Verwenden Sie in Server-Umgebungen nicht die Einstellung “Prompt” (Eingabeaufforderung). Diese Einstellung führt zu einem “hängenden” Server, weil kein Client dazu in der Lage sein wird, die Eingabeaufforderung zu beantworten.

In der Windows-Umgebung werden Cookies automatisch von der Windows API verarbeitet, dies bedeutet, dass wenn Cookies im Browser eingeschaltet werden, alle eingehenden Cookies gespeichert und mit der nächsten Anforderung automatisch übermittelt werden.

Anmerkung:

In einer UNIX- oder OpenVMS-Umgebung müssen die folgenden Profil-Parameter beachtet werden: NOPROX, PROXPORT, PROX.

Informationen über diese Parameter entnehmen Sie den Profilparameter-Beschreibungen im *Natural Referenzhandbuch*.

operand1

Operand1 ist die URL, um auf ein Dokument zuzugreifen.

Die nachstehenden Informationen sind nur gültig, wenn *operand1* mit "http://" oder in einer Windows-Umgebung auch mit "https://" beginnt.

operand2

Operand2 ist der Name des Benutzers, der für die Anforderung verwendet wird.

operand3

Operand3 ist das Passwort des Benutzers, das für die Anforderung verwendet wird.

operand4/5

Operand4 ist der Name einer mit dieser Anforderung gesendeten HEADER-Variable.

Operand5 ist der Wert einer mit dieser Anforderung gesendeten HEADER-Variable.

Anmerkung:

Operand4 und operand5 können nur in Verbindung miteinander verwendet werden.

Beschränkungen

Header-Namen für Operand4

CR/LF und ":" dürfen nicht in Header-Namen vorkommen. Dies wird vom Statement REQUEST DOCUMENT aber nicht überprüft. Gültige Header-Namen finden Sie in den HTTP-Spezifikationen. Aus Gründen der Kompatibilität mit der Web-Schnittstelle können Header-Namen mit "_" anstatt mit "-" geschrieben werden. Intern wird "_" durch "-" ersetzt.

Header-Wert für Operand5

CR/LF darf nicht in Header-Namen vorkommen. Dies wird vom Statement REQUEST DOCUMENT aber nicht überprüft. Gültige Header-Werte und -Formate finden Sie in den HTTP-Spezifikationen.

Allgemeine Informationen

Für eine HTTP-Anforderung sind einige Header erforderlich, z.B. Request Method (Anforderungsmethode) oder Content-Type (Inhaltstyp).

Diese Header werden automatisch generiert, und zwar abhängig von den mit dem REQUEST DOCUMENT-Statement angegebenen Parametern

Automatisch generierte Header (operand 4/5)

Request-Method (Anforderungsmethode)

Die folgenden Werte werden für *operand5* unterstützt:

- HEAD
- POST
- GET
- PUT

Die folgende Tabelle zeigt die automatische Berechnung von Request-Method in Abhängigkeit von den angegebenen Operanden:

Anforderungsmethode	Operand	HEAD	POST	GET	PUT
WITH HEADER	4/5	optional	optional	optional	optional
WITH DATA	7/8	ohne Angabe	ist anzugeben	ohne Angabe	nur mit der Option ALL (<i>operand6</i>)
RETURN HEADER	9 bis 11	ist anzugeben	optional	optional	optional
RETURN PAGE	12	ohne Angabe	ist anzugeben	ist anzugeben	optional

Content-Type (Inhaltstyp)

Wenn POST die Anforderungsmethode ist, muss ein Content-Type-Header mit der HTTP-Anforderung übermittelt werden. Wenn explizit kein Content-Type gesetzt ist, ist der automatisch generierte Wert von *operand5* wie folgt:

```
application/x-www-form-urlencoded
```

Anmerkung:

Es ist möglich, die automatisch generierten Header zu überschreiben. Natural überprüft sie nicht auf Fehler. Unerwartete Fehler können auftreten.

operand6

Operand6 ist ein vollständiges Dokument, das übermittelt wird. Dieser Wert ist für die HTTP-Anforderungsmethode PUT erforderlich.

operand7/8

Operand7 ist der Name einer mit dieser Anforderung übermittelten DATA-Variablen. Dieser Wert ist für den HTTP-Anforderungstyp POST erforderlich (Dekodieren der URL nötig – insbesondere “&”, “=” und “%”).

Operand8 ist der Wert einer mit dieser Anforderung übermittelten DATA-Variablen. Dieser Wert ist für den HTTP-Anforderungstyp POST erforderlich (Dekodieren der URL nötig – insbesondere “&”, “=” und “%”).

Anmerkung:

Operand7 und operand8 können nur in Verbindung miteinander verwendet werden.

Beschränkung

Wird *operand 7/8* angegeben und standardmäßig die Kommunikation “http://” oder “https://” eingesetzt, wird die Anforderungsmethode **POST** (siehe folgende Tabelle) mit Content-Type **application/x-www-form-urlencoded** verwendet.

Im Verlauf der Anforderung werden die *operands 7/8* durch “=”- und “&”-Zeichen voneinander getrennt. Deshalb dürfen zumindest “=” und “&” nicht in Header-Namen vorkommen, und wegen der URL-Kodierung, auch nicht das “%”-Zeichen. Diese Zeichen werden als “unsicher” angesehen und müssen folgendermaßen kodiert werden:

Zeichen	Syntax für URL-Kodierung
%	%25
&	%26
=	%3D

Allgemeine Anmerkung zur URL-Kodierung

Beim Übermitteln von POST-Daten mit Content-Type “application/x-www-form-urlencoded” müssen bestimmte Zeichen mittels URL-Kodierung dargestellt werden, was bedeutet, daß das Zeichen durch %hexadezimalen Zeichencode ersetzt wird. Die vollständigen Einzelheiten darüber, wann und warum eine URL-Kodierung erforderlich ist, sind in RFC 1630, RFC 1738 und RFC 1808 erörtert.

Einige grundsätzliche Einzelheiten sind dort angegeben. Alle Nicht-ASCII-Zeichen (d.h. gültige ISO 8859/1-Zeichen, die nicht auch ASCII-Zeichen sind) müssen URL-kodiert sein. Zum Beispiel würde die Datei köln.html in einer URL als k%F6ln.html erscheinen.

Einige Zeichen werden als “unsicher” betrachtet, wenn Web-Seiten über Email angefordert werden.

Diese sind:

Zeichen	Syntax für URL-Kodierung
Tab-Zeichen	%09
Leerzeichen	%20
[%5B
\	(%5C)
]	(%5D)
^	(%5E)
‘	(%60)
{	(%7B)
	(%7C)
}	(%7D)
~	(%7E)

Beim Schreiben von URLs sollten Sie diese Zeichen URL-kodieren.

Einige Zeichen haben im Rahmen von URLs besondere Bedeutungen, wie z.B. der Doppelpunkt (:), der das URL-Schema vom Rest der URL trennt, der //, der anzeigt, dass die URL in Übereinstimmung mit der allgemein üblichen Syntax für Internet-Schemata ist, und das Prozentzeichen (%).

Wenn diese Zeichen als Bestandteile von Dateinamen erscheinen, müssen sie URL-kodiert sein, damit sie sich von ihrer Spezialbedeutung im Rahmen von URLs abheben (dies ist eine Vereinfachung, volle Einzelheiten erhalten sie in den RFCs).

Diese Zeichen sind folgende:

Zeichen	Syntax für URL-Kodierung
”	(%22)
#	(%23)
%	(%25)
&	(%26)
+	(%2B)
,	(%2C)
/	(%2F)
:	(%3A)
<	(%3C)
=	(%3D)
>	(%3E)
?	(%3F)
@	(%40)

operand9

Operand9 enthält alle mit der HTTP-Antwort gelieferten Header-Werte

Die erste Zeile enthält die Status-Informationen und alle folgenden Zeilen enthalten die Header als Namen- und Wertepaare. Die Namen enden immer auf einen (:) und die Werte auf einen (CR). (Intern werden alle “CR/LF”s in “CR”s umgesetzt).

operand10/11

Operand10 ist der Name von einem mit dieser Anforderung empfangenen HEADER. Der HEADER ist für HTTP erforderlich.

Operand11 ist der Wert von einem mit dieser Anforderung empfangenen HEADER. Der HEADER ist für HTTP erforderlich.

Anmerkung:

Operand10 und operand11 können nur in Verbindung miteinander verwendet werden.

Header-Name für Operand10 zurückgeben

Aus Gründen der Kompatibilität mit der Web-Schnittstelle können Header-Namen mit “_” anstatt mit “-” geschrieben werden. Intern wird “_” ersetzt durch “-”.

Wenn operand10 eine leere Zeichenkette ist, werden die Status-Informationen zurückgegeben.

```
HTTP/1.0 200 OK
```

operand12

Operand12 ist das für diese Anforderung zurückgegebene Dokument.

operand13

Operand13 ist die Antwort-Nummer der Anforderung (z.B. 200).

Übersicht über die Antwort-Nummern – für HTTP/HTTP-Anforderungen

Status	Wert	Antwort
STATUS CONTINUE	100	OK zur Fortsetzung der Anforderung
STATUS SWITCH_PROTOCOLS	101	Server hat Protokolle gewechselt, um Header auf den neuesten Stand zu bringen
STATUS OK	200	Anforderung erfüllt
STATUS CREATED	201	Objekt angelegt, Grund = neue URL
STATUS ACCEPTED	202	Async beendet (TBS)
STATUS PARTIAL	203	Teilweise Beendigung
STATUS NO_CONTENT	204	Keine Infos zurückzugeben
STATUS RESET_CONTENT	205	Anforderung beendet, Formular wurde gelöscht
STATUS PARTIAL_CONTENT	206	Teilweises GET beendet
STATUS AMBIGUOUS	300	Server kann nicht entscheiden, was zurückzugeben ist
STATUS MOVED	301	Objekt für längere Zeit verschoben
STATUS REDIRECT	302	Objekt zeitweilig verschoben
STATUS REDIRECT_METHOD	303	Umleitung ohne neue Zugriffsmethode
STATUS NOT_MODIFIED	304	If-modified-since (Änderung seit) nicht geändert
STATUS USE_PROXY	305	Umleitung zum Proxy, Location-Header gibt zu verwendenden Proxy an
STATUS REDIRECT_KEEP_VERB	307	HTTP/1.1: Verb beibehalten
STATUS BAD_REQUEST	400	Ungültige Syntax
STATUS DENIED	401	Zugriff verweigert
STATUS PAYMENT_REQ	402	Zahlung erforderlich
STATUS FORBIDDEN	403	Anforderung verboten
STATUS NOT_FOUND	404	Objekt nicht gefunden
STATUS BAD_METHOD	405	Methode ist nicht zulässig
STATUS NONE_ACCEPTABLE	406	Keine Antwort akzeptabel für gefundenen Client
STATUS PROXY_AUTH_REQ	407	Authentifizierung durch ein Proxy erforderlich
STATUS REQUEST_TIMEOUT	408	Zeitüberschreitung beim Warten auf die Anforderung
STATUS CONFLICT	409	Benutzer sollte mit mehr Infos neu ausführen

Status	Wert	Antwort
STATUS GONE	410	Die Ressource ist nicht mehr verfügbar
STATUS LENGTH_REQUIRED	411	Server weigerte sich, die Anforderung ohne Längenangabe zu akzeptieren
STATUS PRECOND_FAILED	412	In Anforderung gegebene Voraussetzung gescheitert
STATUS REQUEST_TOO_LARGE	413	Anforderungselement war zu groß
STATUS URL_TOO_LONG	414	Anforderungs-URL zu lang
STATUS UNSUPPORTED_MEDIA	415	Nicht unterstützter Medien-Typ
STATUS SERVER_ERROR	500	Interner Server-Fehler
STATUS NOT_SUPPORTED	501	“Required” nicht unterstützt
STATUS BAD_GATEWAY	502	Fehler-Antwort vom Gateway empfangen
STATUS SERVICE_UNAVAIL	503	Zeitweilig überlastet
STATUS GATEWAY_TIMEOUT	504	Zeitüberschreitung beim Warten auf Gateway
STATUS VERSION_NOT_SUP	505	HTTP-Version nicht unterstützt

Beschränkungen

Antwort 301 – 303 (Redirection)

Redirection (Umleitung) bedeutet, dass die angeforderte URL umgezogen ist. Als Antwort wird ein RETURN HEADER mit dem Namen LOCATION angezeigt. Dieser Header enthält die URL, wohin die angeforderte Seite umgezogen ist. Eine neue REQUEST DOCUMENT-Anforderung kann verwendet werden, um die umgezogene Seite zu suchen.

HTTP-Browser leiten automatisch auf die neue URL um, aber das Statement REQUEST DOCUMENT führt eine Umleitung nicht automatisch durch.

Antwort 401 (Denied)

Die Antwort “Access Denied” (Zugriff verweigert) bedeutet, dass die angeforderte Seite nur aufgerufen werden kann, wenn eine gültige UserID (Benutzerkennung) und ein gültiges Passwort mit der Anforderung übermittelt wird. Als Antwort wird ein Return Header mit dem Namen WWW-AUTHENTICATE mit dem für diese Anforderung erforderlichen Bereich übermittelt.

HTTP-Browser zeigen normalerweise einen UserID-Passwort-Dialog an, aber im Falle des Statements REQUEST DOCUMENT wird kein Dialog angezeigt.

operand14

Operand14 enthält den Natural-Fehler, wenn die Anforderung nicht ausgeführt werden konnte.

Beispiele:

Anmerkung:

Für dieses Statement gibt es einen Beispiel-Dialog V5-RDOC in der Beispiels-Bibliothek SYSEXV.

Allgemeine Anforderung

```
REQUEST DOCUMENT FROM "http://bolsapl:5555/invoke/sap.demo/handle_RFC_XML_POST"
WITH
  USER #User PASSWORD #Password
  DATA
  NAME 'XMLData'          VALUE #Queryxml
  NAME 'repServerName' VALUE 'NT2'
RETURN
  PAGE #Resultxml
RESPONSE #rc
```

Einfache GET-Anforderung (keine Daten)

```
REQUEST DOCUMENT FROM "http://pcnatweb:8080"
RETURN
  PAGE #Resultxml
RESPONSE #rc
```

Einfache HEAD-Anforderung (keine Rückgabeseite)

```
REQUEST DOCUMENT FROM "http://pcnatweb"
RESPONSE #rc
```

Einfache POST-Anforderung (Voreinstellung)

```
REQUEST DOCUMENT FROM "http://pcnatweb/cgi-bin/nwvcgi.exe/sysweb/nat-env"
WITH
  DATA
  NAME 'XMLData'          VALUE #Queryxml
  NAME 'repServerName' VALUE 'NT2'
RETURN
  PAGE #Resultxml
RESPONSE #rc
```

Einfache PUT-Anforderung (mit DATA ALL)

```
REQUEST DOCUMENT FROM "http://pcnatweb/test.txt"
WITH
  DATA ALL #document
RETURN
  PAGE #Resultxml
RESPONSE #rc
```

RESET

RESET [INITIAL] *operand1...*

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S A G M	A N P I F B D T L C G O	ja	ja

Funktion

Mit dem Statement RESET können Sie den Wert eines oder mehrerer Operanden auf Null oder auf einen im DEFINE DATA-Statement definierten Ausgangswert zurücksetzen.

Wenn *operand1* eine DYNAMISCHE Variable ist, wird sie auf einen Nullwert zurückgesetzt, und zwar mit der Länge, die die Variable zum Zeitpunkt der Ausführung des RESET-Statements hat. Die aktuelle Länge einer DYNAMISCHEN Variable kann mittels der Systemvariable *LENGTH ermittelt werden. Allgemeine Informationen über DYNAMISCHE Variablen finden Sie in Ihrem *Natural Benutzerhandbuch* oder *Natural User's Guide*.

(Im *Reporting Mode* können Sie mit RESET auch eine Variable definieren, vorausgesetzt, das Programm enthält kein DEFINE DATA LOCAL-Statement.)

INITIAL

RESET (ohne INITIAL) setzt jedes angegebene Feld (*operand1*) auf einen Nullwert.

Mit RESET INITIAL werden die angegebenen Felder auf die im für sie im DEFINE DATA-Statement definierten Ausgangswerte (initial values) zurückgesetzt. Wenn für ein Feld kein Ausgangswert definiert ist, wird es auf einen Standard-Ausgangswert (siehe unten) zurückgesetzt.

Wenn Sie RESET INITIAL auf ein Array anwenden, müssen Sie es auf das gesamte Array (wie im DEFINE DATA-Statement definiert) anwenden; RESET INITIAL für einzelne Array-Ausprägungen ist nicht möglich.

RESET INITIAL für Felder, die aus einer Redefinition resultieren, ist ebenfalls nicht möglich. Auf Datenbankfelder ist RESET INITIAL nicht anwendbar.

Auf DYNAMISCHE Variablen ist RESET INITIAL nicht anwendbar.

Standard-Ausgangswerte

Wenn Sie im DEFINE DATA-Statement keine INIT- oder CONST-Angaben machen, wird ein Feld je nach Format mit einem Standard-Ausgangswert initialisiert.

Beispiel

```

/* EXAMPLE 'RSTEX1': RESET
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME (A10)
1 #BINARY (B4) INIT <1>
1 #INTEGER (I4) INIT <5>
1 #NUMERIC (N2) INIT <25>
END-DEFINE
/*****
LIMIT 1
READ EMPLOY-VIEW
/*****
WRITE NOTITLE 'VALUES BEFORE RESET STATEMENT:'
WRITE / '=' NAME '=' #BINARY '=' #INTEGER '=' #NUMERIC
/*****
RESET NAME #BINARY #INTEGER #NUMERIC
WRITE /// 'VALUES AFTER RESET STATEMENT:'
WRITE / '=' NAME '=' #BINARY '=' #INTEGER '=' #NUMERIC
/*****
RESET INITIAL #BINARY #INTEGER #NUMERIC
WRITE /// 'VALUES AFTER RESET INITIAL STATEMENT:'
WRITE / '=' NAME '=' #BINARY '=' #INTEGER '=' #NUMERIC
/*****
END-READ
END

```

VALUES BEFORE RESET STATEMENT:

NAME: MORENO #BINARY: 00000001 #INTEGER: 5 #NUMERIC: 25

VALUES AFTER RESET STATEMENT:

NAME: #BINARY: 00000000 #INTEGER: 0 #NUMERIC: 0

VALUES AFTER RESET INITIAL STATEMENT:

NAME: #BINARY: 00000001 #INTEGER: 5 #NUMERIC: 25

RETRY

Anmerkung:

Dieses Statement kann nur beim Zugriff auf Adabas-Datenbanken verwendet werden.

RETRY

Verwandte Statements

FIND, READ.

Funktion

Das Statement RETRY wird in einem ON ERROR-Statement-Block (siehe ON ERROR-Statement) verwendet. Es dient dazu, erneut zu versuchen, auf einen Datensatz zuzugreifen, auf den bereits ein anderer Benutzer zugegriffen hat und der sich daher im “Hold”-Status befindet.

Befindet sich ein Datensatz im “Hold”, gibt Natural die Fehlermeldung 3145 aus. Siehe auch Session-Parameter WH im *Natural Referenzhandbuch*.

Das RETRY-Statement muß in dem Objekt stehen, das die Fehlermeldung 3145 verursacht.

Weitere Informationen zur Hold-Logik finden Sie im Kapitel **Datenbankzugriffe** des *Natural Leitfadens zur Programmierung*.

Beispiel

```

/* EXAMPLE 'RTYEX1S': RETRY (STRUCTURED MODE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
1 #RETRY (A1) INIT <' '>
END-DEFINE
/*****
FIND EMPLOY-VIEW WITH NAME = 'ALDEN'
DELETE
END TRANSACTION
/*****
ON ERROR
IF *ERROR-NR = 3145
  INPUT NO ERASE 10/1
    'RECORD IS IN HOLD' /
    'DO YOU WISH TO RETRY ' /
    #RETRY '(Y)ES OR (N)O '
  IF #RETRY = 'Y'
    RETRY
  ELSE
    STOP
  END-IF
END-IF
END-ERROR
/*****
AT END OF DATA
WRITE NOTITLE *NUMBER 'RECORDS DELETED'
END-ENDDATA
END-FIND
/*****
END

```

Äquivalentes Reporting-Mode-Beispiel: siehe Programm RTYEX1R in Library SYSEXRM.

RUN

RUN [REPEAT] operand1 [operand2 [(parameter)]] ...⁴⁰

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	ja	nein
Operand2	C S A G	A N P I F B D T L G	ja	nein

Funktion

Das Statement RUN dient dazu, ein Natural-Source-Programm aus der Natural-Systemdatei zu lesen und es dann auszuführen.

REPEAT

Mit RUN REPEAT wird ein Programm vollständig ausgeführt, ohne daß der Benutzer zwischendurch auf etwaigen (durch INPUT-Statements ausgegebenen) Ausgabeschirmen durch eine Eingabe reagieren müßte.

Diese Option kann eingesetzt werden, wenn ein Programm mehrere Schirme mit Informationen ausgibt, bei denen es nicht erforderlich ist, daß der Benutzer auf jeden ausgegebenen Schirm reagiert.

Programmname (*operand1*)

Der Name des auszuführenden Programms kann als alphanumerische Konstante oder als Inhalt einer alphanumerischen Variablen angegeben werden. Wird eine Variable verwendet, so muß sie 8 Stellen lang sein.

Das Programm kann entweder in der aktuellen Library oder in einer Steplib (Standard-Steplib ist "SYSTEM") gespeichert sein. Wird es dort nicht gefunden, gibt Natural eine Fehlermeldung aus.

Das ausgeführte Programm wird in den Arbeitsbereich des Programm-Editors gelesen und überlagert dabei den vorherigen Inhalt des Arbeitsbereichs.

Parameter (*operand2*)

Mit dem RUN-Statement können Parameter an das Programm, das ausgeführt werden soll, übergeben werden. Die Parameter können mit beliebigem Format definiert werden; sie werden automatisch in Formate umgesetzt, die zu den entsprechenden INPUT-Feldern passen. Alle angegebenen Parameter werden oben auf dem Natural-Stack abgelegt.

Parameter können von einem INPUT-Statement gelesen werden. Das erste INPUT-Statement fügt alle Parameter in die im INPUT-Statement angegebenen Felder ein. Bei numerischen Feldern muß der Vorzeichen-Parameter auf SG=ON gesetzt werden.

Werden mehr Parameter übergeben als INPUT-Felder vorhanden sind, so werden überschüssige Parameter ignoriert. Die Anzahl der Parameter kann über die Systemvariable *DATA ermittelt werden.

Anmerkung:

Wenn operand2 eine Zeitvariable (Format T) ist, wird nur die Zeitkomponente des Variableninhalts übergeben, aber nicht die Datumskomponente.

parameter

Wenn *operand2* eine Datumsvariable ist, können Sie den Session-Parameter DF als *parameter* für diese Variable angeben. Der Session-Parameter DF ist im *Natural Referenzhandbuch* beschrieben.

Dynamische Sourcecode-Generierung und -Ausführung

Das RUN-Statement kann dazu verwendet werden, ein Programm zu kompilieren und auszuführen, dessen Source ganz oder teilweise dynamisch erstellt wird.

Dynamische Sourcecode-Generierung erfolgt, indem man Sourcetext in globalen Variablen *Natural* unterbringt, und diese Variablen dadurch referenziert, daß man im Sourcecode das erste Zeichen der Variablennamen “+” jeweils durch ein “&” ersetzt. Wird das Programm mit RUN aufgerufen, so wird der Inhalt der globalen Variablen als Sourcecode interpretiert.

Eine globale Variable mit Index darf nicht in einem mit RUN ausgeführten Programm verwendet werden.

Eine globale Variable darf keinen Kommentar und kein INCLUDE-Statement enthalten.

Beispiel

Programm mit RUN-Statement:

```

/* EXAMPLE 'RUNEX1': RUN (WITH DYNAMIC SOURCE PROGRAM CREATION)
/*****
/* GLOBAL AREA 'GDA1' CONTAINS '+CRITERIA'
/* +CRITERIA (A80)
/* PROGRAM CREATING SOURCE
DEFINE DATA GLOBAL USING GDA1
LOCAL
1 #NAME (A20)
1 #CITY (A20)
END-DEFINE
/*****
INPUT 'Please specify the search values:' /
      'Name:' #NAME /
      'City:' #CITY
RESET +CRITERIA
/*****
IF #NAME = ' ' AND #CITY = ' '
    REINPUT 'Enter at least 1 value'
END-IF
/*****
IF #NAME NE ' '
    COMPRESS 'NAME' ' '='' #NAME '''' INTO +CRITERIA LEAVING NO
END-IF
IF #CITY NE ' '
    IF +CRITERIA NE ' '
        COMPRESS +CRITERIA 'AND' INTO +CRITERIA
    END-IF
    COMPRESS +CRITERIA ' CITY =' ' ' #CITY '''' INTO +CRITERIA
END-IF
/*****
RUN 'FIND-EMP'
/*****
END

```

Programm FIND-EMP, das per RUN-Statement ausgeführt wird:

```

/* PROGRAM EXECUTED WITH RUN STATEMENT ('FIND-EMP')
/*****
DEFINE DATA GLOBAL USING GDA1
LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
2 NAME
2 CITY
END-DEFINE
/*****
FIND NUMBER EMPLOY-VIEW WITH &CRITERIA
RETAIN AS 'EMP-SET'

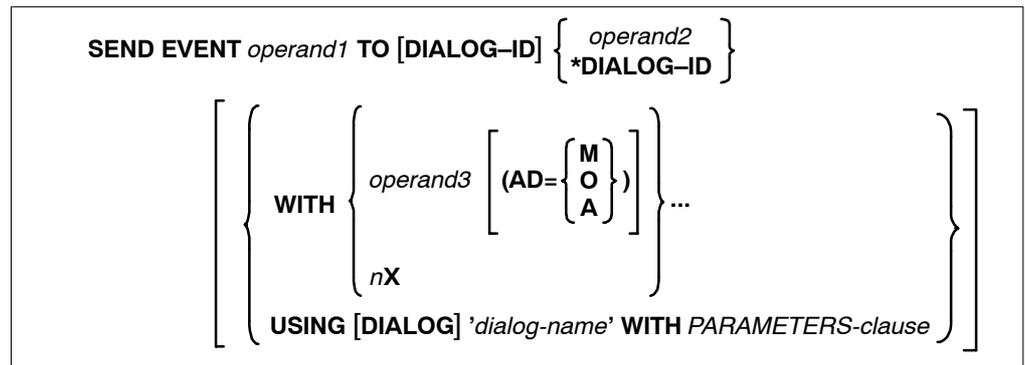
    DISPLAY *NUMBER
END

```

SEND EVENT

Anmerkung:

Dieses Statement ist nur unter Windows und Windows NT verfügbar.



Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	ja	nein
Operand2	S	I	ja	nein
Operand3	C S A	A N P I F B D T L C G O	ja	nein

Funktion

Mit diesem Statement lösen Sie einen benutzerdefinierten Event in einer Natural-Anwendung aus.

Operanden

Operand1 ist der Name des Events, der ausgelöst werden soll.

Operand2 ist die Kennung (ID) des Dialogs, der den Event erhält. *Operand2* muß mit Format/Länge 14 definiert werden.

AD=

Wenn *operand3* eine Variable ist, können Sie sie wie folgt kennzeichnen:

AD=O	nicht modifizierbar
AD=M	modifizierbar
AD=A	nur für Eingabe

Standardmäßig gilt AD=M.

Wenn *operand3* eine Konstante ist, kann *operand3* nicht explizit angegeben werden. Für Konstanten gilt immer AD=O.

AD=M

Standardmäßig kann der übergebene Wert eines Parameters im Dialog geändert werden und der geänderte Wert an das aufrufende Objekt zurückgegeben werden, wo er den ursprünglichen Wert überschreibt.

Ausnahme: Bei einem Feld, das in der Parameter Data Area des Dialogs mit BY VALUE definiert ist, wird kein Wert zurückgegeben.

AD=O

Wenn Sie einen Parameter mit AD=O kennzeichnen, kann der übergebene Wert im Dialog geändert werden, aber der geänderte Wert nicht an das aufrufende Objekt zurückgegeben werden; d.h. im aufrufenden Objekt behält das Feld seinen ursprünglichen Wert.

Anmerkung:

Intern wird AD=O wie BY VALUE verarbeitet (vgl. Abschnitt Parameter-Data-Definition des DEFINE DATA-Statements).

AD=A

Kennzeichnen Sie einen Parameter mit AD=A, wird sein Wert nicht *an* den Dialog übergeben, sondern er erhält einen Wert *vom* Dialog. Bevor der Event gesendet wird, werden "AD=A"-Felder auf Leerwert zurückgesetzt.

Bei einem Feld, das in der Parameter Data Area des Dialogs mit BY VALUE definiert ist, kann das aufrufende Objekt keinen Wert erhalten. Hier bewirkt AD=A lediglich, daß das Feld vor dem Senden des Events auf Leerwert zurückgesetzt wird.

Übergabe von Parameter an den Dialog

Es ist möglich, Parameter an den Dialog zu übergeben.

Als *operand3* geben Sie die Parameter an, die an den Dialog übergeben werden sollen.

Mit der *PARAMETERS-clause* können Parameter selektiv übergeben werden.

nX

Mit der Notation *nX* können Sie angeben, daß die nächsten *n* Parameter übersprungen werden sollen (z.B. 1X, um den nächsten Parameter zu überspringen, oder 3X, um die nächsten drei Parameter zu überspringen), dies bedeutet, daß für die nächsten *n* Parameter keine Werte an den Dialog übergeben werden.

Ein zu überspringender Parameter muß mit dem Schlüsselwort **OPTIONAL** im Statement **DEFINE DATA PARAMETER** des Dialogs definiert werden. **OPTIONAL** bedeutet, daß ein Wert vom aufrufenden Objekt an einen solchen Parameter übergeben werden kann — aber nicht unbedingt muß.

PARAMETERS-clause

```
PARAMETERS {parameter-name = operand3} ... END-PARAMETERS
```

Anmerkung:

Sie können die PARAMETERS-clause nur verwenden, wenn der angegebene Zieldialog (dialog-name) katalogisiert ist.

Dialog-name ist der Name des Dialogs, der den Event erhält.

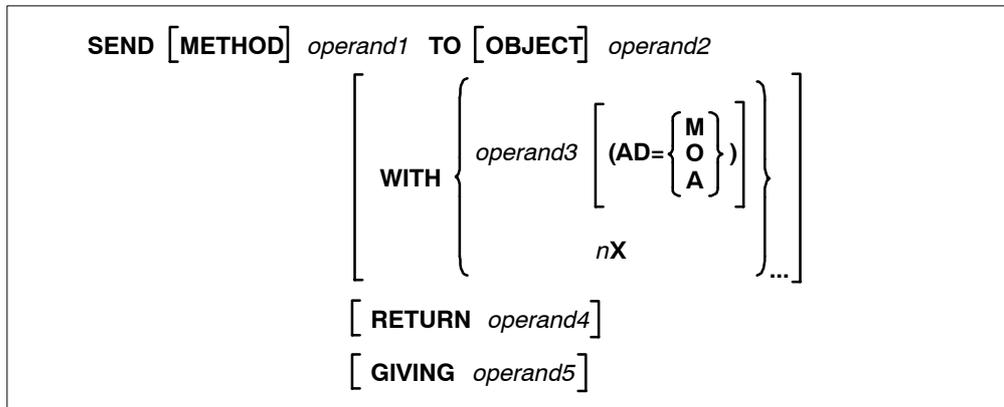
Anmerkung:

Wenn der Wert eines mit AD=O gekennzeichneten, "by reference" übergebenen Parameters in einem Dialog geändert wird, führt dies zu einem Laufzeitfehler.

Weitere Informationen und Beispiele

Siehe Kapitel **Event-Driven Programming Techniques** im *Natural User's Guide for Windows*.

SEND METHOD



Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	ja	nein
Operand2	S	O	nein	nein
Operand3	C S A G	A N P I F B D T L C G O	ja	nein
Operand4	S A	A N P I F B D T L C G O	ja	nein
Operand5	S N	I	ja	nein

Format C und G kann nur an Methoden lokaler Klassen übergeben werden. Weitere Informationen entnehmen Sie dem Abschnitt **Lokale Klassen**.

Funktion

Das SEND METHOD-Statement dient dazu, eine bestimmte Methode eines Objekts aufzurufen.

Anmerkung:

Optionale Parameter (nX-Notation) stehen mit der Version 4.1.1 und allen nachfolgenden Releases zur Verfügung. Die AD-Option steht mit der Version 4.1.2 und allen nachfolgenden Releases zur Verfügung.

Method-Name — *operand1*

Operand1 ist der Name einer Methode, die vom in *operand2* angegebenen Objekt unterstützt wird.

Da die Methoden-Namen in unterschiedlichen Interfaces (Schnittstellen) einer Klasse identisch sein können, kann der Methoden-Name in *operand1* auch mit dem Interface-Namen versehen werden, um Mehrdeutigkeiten zu vermeiden.

Im folgenden Beispiel hat das Objekt #O3 eine Interface-Iteration mit der Methode "Start". Es gelten die folgenden Statements:

```
* Specifying only the method name.  
SEND "Start" TO #O3  
* Qualifying the method name with the interface name.  
SEND "Iterate.Start" TO #O3
```

Wenn kein Interface-Name angegeben wird, sucht Natural den Methoden-Namen in allen Interfaces der Klasse. Wenn der Methoden-Name in mehr als einem Interface gefunden wurde, tritt ein Laufzeitfehler auf.

Object Handle — *operand2*

Die Handle des Objekts, an das der Aufruf der Methode gesendet werden soll.

Operand2 muß als Objekt-Handle (HANDLE OF OBJECT) definiert werden. Das Objekt muß bereits vorhanden sein.

Um eine Methode des aktuellen Objekts innerhalb einer Methode aufzurufen, verwenden Sie die Systemvariable *THIS-OBJECT.

Parameter — *operand3*

Als *operand3* können Sie Parameter angeben, die methodenspezifisch sind.

Im folgenden Beispiel hat das Objekt #O3 die Methode “PositionTo” mit dem Parameter “Pos”. Die Methode wird wie folgt aufgerufen:

```
SEND "PositionTo" TO #O3 WITH Pos
```

Methoden können optionale Parameter haben. Optionale Parameter brauchen nicht angegeben zu werden, wenn die Methode aufgerufen wird. Um einen optionalen Parameter wegzulassen, verwenden Sie den Platzhalter 1X. Um *n* optionale Parameter wegzulassen, verwenden Sie den Platzhalter *nX*.

Im folgenden Beispiel hat die Methode “SetAddress” des Objekts “#O4” die Parameter “FirstName” (Vorname), “MiddleInitial” (Mittlere Initiale), “LastName” (Nachname), “Street” (Straße) und “City” (Stadt), wobei “MiddleInitial”, “Street” und “City” optional sind. Es gelten die folgenden Statements:

```
* Specifying all parameters.  
SEND "SetAddress" TO #O4 WITH FirstName MiddleInitial LastName Street City  
* Omitting one optional parameter.  
SEND "SetAddress" TO #O4 WITH FirstName 1X LastName Street City  
* Omitting all optional parameters.  
SEND "SetAddress" TO #O4 WITH FirstName 1X LastName 2X
```

Wenn ein nicht optionaler (zwingender) Parameter weggelassen wird, so führt dies zu einem Laufzeitfehler.

AD=

Wenn *operand3* eine Variable ist, können Sie sie wie folgt kennzeichnen:

AD=O	nicht modifizierbar
AD=M	modifizierbar
AD=A	nur für Eingabe

Standardmäßig gilt AD=M.

Wenn *operand3* eine Konstante ist, kann AD nicht explizit angegeben werden. Für Konstanten gilt immer AD=O.

AD=M

Dies ist die Voreinstellung, der übergebene Wert eines Parameters kann in der Methode geändert werden, und der geänderte Wert wird an den Aufrufer zurückgegeben, wo er den ursprünglichen Wert überschreibt.

Wenn eine Methode in Natural implementiert ist, und der Parameter in der Parameter Data Area der Methode mit BY VALUE definiert wird, wird kein Wert zurückgegeben.

Wenn eine Methode nicht in Natural implementiert ist, ist das Verhalten von der Implementierung der Methode abhängig. Der Parameter wird dann BY REFERENCE übergeben. Ob die externe Komponente einen Parameter der Art "by reference" oder "by value" akzeptiert, ist in der Dokumentation der externen Komponente beschrieben. Sie kann auch im Natural Component Browser angesehen werden.

AD=O

Wenn Sie einen Parameter mit AD=O kennzeichnen, kann der übergebene Wert in der Methode geändert werden, aber der geänderte Wert wird nicht an das aufrufende Objekt zurückgegeben. Der Aufruf behält seinen ursprünglichen Wert bei.

Wenn eine Methode in Natural implementiert wird, wird der Parameter so behandelt, wie er in der Parameter Data Area der Methode BY VALUE definiert war (siehe Abschnitt *PARAMETER-clause* in der Beschreibung des INTERFACE-Statements).

Wenn eine Methode in Natural nicht implementiert ist, ist das Verhalten von der Implementierung der Methode abhängig. Der Parameter wird dann BY VALUE übergeben. Ob die externe Komponente einen Parameter der Art "by reference" oder "by value" akzeptiert, ist in der Dokumentation der externen Komponente beschrieben. Sie kann auch im Natural Component Browser angesehen werden.

AD=A

Wenn Sie einen Parameter mit AD=A kennzeichnen, wird sein Wert nicht an die Methode übergeben, aber er erhält einen Wert von der Methode. "AD=A"-Felder werden auf Leerzeichen zurückgesetzt, bevor die Methode aufgerufen wird.

Bei einem Feld, das in der Parameter Data Area der Methode mit BY VALUE definiert ist, kann der Aufruf keinen Wert erhalten. Hier bewirkt AD=A lediglich, daß das Feld vor dem Aufruf der Methode auf Leerzeichen zurückgesetzt wird.

Wenn in Natural keine Methode implementiert ist, ist das Verhalten von der Implementierung der Methode abhängig. Der Parameter wird dann als initialisierte Variante übergeben. Ob die externe Komponente auch einen Wert zurückgeben kann, ist in der Dokumentation der externen Komponente beschrieben. Sie kann auch im Natural Component Browser angesehen werden.

Parameter – nX

Diese Notation steht auf Großrechnern nicht zur Verfügung.

Mit der Notation nX können Sie angeben, daß die nächsten n Parameter übersprungen werden sollen (zum Beispiel 1X, um den nächsten Parameter zu überspringen, oder 3X, um die nächsten drei Parameter zu überspringen). Dies bedeutet, daß für die nächsten n Parameter an die Methode keine Werte übergeben werden.

Bei einer in Natural implementierten Methode muß ein zu überspringender Parameter mit dem Schlüsselwort OPTIONAL im DEFINE DATA PARAMETER-Statement des Subprogramms der Methode definiert sein. OPTIONAL bedeutet, daß ein Wert vom aufrufenden Objekt an einen solchen Parameter übergeben werden kann – aber nicht unbedingt muß.

RETURN — *operand4*

Wenn die RETURN-Klausel weggelassen wird, und die Methode einen Rückgabewert hat, wird der Rückgabewert nicht berücksichtigt.

Wenn die RETURN-Klausel angegeben wird, enthält *operand4* den Rückgabewert der Methode. Wenn die Ausführung der Methode ohne Erfolg abgebrochen wird, wird *operand4* auf seinen ursprünglichen Wert zurückgesetzt.

Anmerkungen:

Bei in Natural geschriebenen Klassen wird der Rückgabewert einer Methode durch Eingabe eines zusätzlichen Parameters in der Parameter Data Area der Methode und durch Kennzeichnung mit 'BY VALUE RESULT' definiert. Weitere Informationen siehe Abschnitt PARAMETER-clause.

Deshalb enthält die Parameter Data Area einer Methode, die in Natural geschrieben ist, und die einen Rückgabewert hat, neben den Methoden-Parametern immer ein zusätzliches Feld. Dies ist zu berücksichtigen, wenn Sie eine Methode einer in Natural geschriebenen Klasse aufrufen, und die Parameter Data Area der Methode im SEND-Statement verwenden möchten.

GIVING — *operand5*

Wenn die GIVING-Klausel nicht angegeben wird, wird die Natural-Laufzeitfehlerverarbeitung angestoßen, wenn ein Fehler auftritt.

Wenn die GIVING-Klausel angegeben wird, enthält *operand5* die Natural-Meldungsnummer, wenn ein Fehler aufgetreten ist, oder Null, wenn kein Fehler aufgetreten ist.

SEPARATE

```

SEPARATE { operand1
           SUBSTRING (operand1,operand2,operand3) }
           [LEFT [JUSTIFIED]] INTO operand4...
           [ IGNORE
             REMAINDER operand5 ]
           [ WITH [RETAINED] { [ANY] DELIMITERS
                               INPUT DELIMITERS
                               DELIMITERS operand6 } ]
           [[GIVING] NUMBER [IN] operand7 ]

```

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	ja	nein
Operand2	C S	N P I	ja	nein
Operand3	C S	N P I	ja	nein
Operand4	S A G	A	ja	ja
Operand5	S	A	ja	ja
Operand6	C S	A	ja	nein
Operand7	S	N P I	ja	ja

Verwandte Statements

COMPRESS, EXAMINE.

Funktion

Das Statement SEPARATE dient dazu, den Inhalt eines alphanumerischen Operanden auf zwei oder mehr alphanumerische Operanden (oder auf mehrere Ausprägungen eines alphanumerischen Arrays) zu verteilen.

Ausgangsoperand (*operand1*)

Operand1 ist die alphanumerische Konstante oder Variable, deren Inhalt aufgeteilt werden soll.

Nachgestellte Leerzeichen in *operand1* werden entfernt, bevor der Wert verarbeitet wird (auch wenn das Leerzeichen als Delimiterzeichen verwendet wird; vgl. DELIMITER-Option).

SUBSTRING

Normalerweise wird der ganze Inhalt des Feldes aufgeteilt, und zwar vom Anfang des Feldes an. — Die Option SUBSTRING ermöglicht es Ihnen, nur einen bestimmten Teil des Feldes aufzuteilen. In der SUBSTRING-Klausel geben Sie nach dem Feldnamen (*operand1*) zunächst die erste Stelle (*operand2*) und dann die Länge (*operand3*) des Feldteils an, der aufgeteilt werden soll. Wenn z.B. ein Feld #A den Wert “CONTRAPTION” enthält, würde SUBSTRING(#A,5,3) den Wert “RAP” enthalten.

Anmerkung:

Wenn Sie operand2 weglassen, wird ab Anfang des Feldes aufgeteilt. Wenn Sie operand3 weglassen, wird ab der angegebenen Stelle (operand2) bis zum Ende des Feldes aufgeteilt.

LEFT JUSTIFIED

Diese Option bewirkt, daß den aufgeteilten Feldwertteilen vorangestellte Leerzeichen aus den Zieloperanden entfernt werden.

Zieloperand (*operand4*)

Operand4 sind die Zieloperanden, die die Teile des Ausgangsoperanden aufnehmen sollen. Wird als Zieloperand ein Array verwendet, wird es Ausprägung für Ausprägung mit den übertragenen Feldwertteilen gefüllt.

Die Anzahl der Zieloperanden entspricht der Anzahl der Delimiterzeichen (einschließlich nachgestellter Delimiterzeichen) in *operand1*, plus 1.

Ist *operand4* eine DYNAMISCHE Variable, kann deren Länge mit der SEPARATE-Operation geändert werden. Die aktuelle Länge einer DYNAMISCHEN Variable kann mittels der Systemvariable *LENGTH ermittelt werden. Allgemeine Informationen zu DYNAMISCHEN Variablen finden Sie in Ihrem *Natural Benutzerhandbuch* oder in Ihrem *Natural User's Guide*.

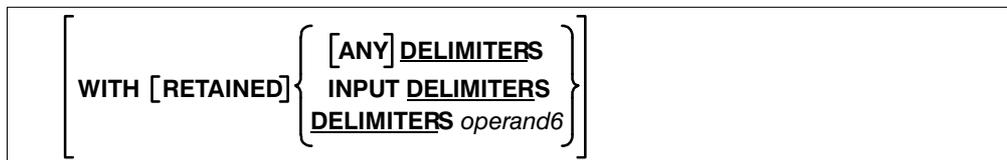
IGNORE/REMAINDER

Wenn Sie nicht genug Zieloperanden angeben, um alle Feldwertteile aufzunehmen, erhalten Sie eine entsprechende Fehlermeldung.

Um dies zu vermeiden, haben Sie zwei Möglichkeiten:

- Wenn Sie IGNORE angeben, ignoriert Natural es, falls nicht genügend Zieloperanden zur Aufnahme des Ausgangswertes vorhanden sind.
- Wenn Sie REMAINDER *operand5* angeben, wird der Teil des Ausgangswertes, für den keine Zieloperanden mehr zur Verfügung stehen, in *operand5* gestellt. Den Inhalt von *operand5* können Sie dann weiter verarbeiten, zum Beispiel in einem weiteren SEPARATE-Statement.

DELIMITER-Option



Delimiterzeichen innerhalb von *operand1* bestimmen die Stellen, an denen der Wert geteilt werden soll.

- Falls Sie die DELIMITER-Option nicht angeben (oder WITH ANY DELIMITER angeben), wird jedes Leerzeichen sowie jedes Zeichen, das weder ein Buchstabe noch eine Ziffer ist, als Delimiterzeichen interpretiert.
- WITH INPUT DELIMITER bedeutet, daß das mit dem Session-Parameter ID definierte Zeichen sowie das Leerzeichen als Delimiter gelten.
- WITH DELIMITER *operand6* bedeutet, daß jedes der angegebenen Zeichen (*operand6*) als Delimiterzeichen interpretiert wird. Wenn *operand6* nachgestellte Leerzeichen enthält, werden diese ignoriert.

WITH RETAINED DELIMITERS

Normalerweise werden die Delimiterzeichen selbst nicht mit in die Zieloperanden übertragen.

Wenn Sie allerdings RETAINED angeben, wird jeder Delimiter (d.h. entweder ID-Zeichen und Leerzeichen oder die mit *operand6* angegebenen Zeichen) ebenfalls in einen Zieloperanden übertragen.

Beispiel:

Das folgende SEPARATE-Statement würde "150" nach #B, "+" nach #C und "30" nach #D übertragen:

```
...
MOVE '150+30' TO #A
SEPARATE #A INTO #B #C #D WITH RETAINED DELIMITER '+'
...
```

GIVING NUMBER

Diese Option bewirkt, daß die Anzahl der Zieloperanden, die mit einem Wert gefüllt wurden (einschließlich der mit Leerzeichen gefüllten), in *operand7* ausgegeben wird. Die Anzahl, die Sie erhalten, errechnet sich aus der Anzahl der Delimiterzeichen plus 1.

Wenn Sie die IGNORE-Option verwenden, enthält *operand7* maximal die Anzahl der Zieloperanden (*operand4*).

Wenn Sie die REMAINDER-Option verwenden, enthält *operand7* maximal die Anzahl der Zieloperanden (*operand4*) plus *operand5*.

Beispiel 1

```

/* EXAMPLE 'SEPEX1': SEPARATE
/*****
DEFINE DATA LOCAL
1 #TEXT1 (A6) INIT <'AAABBB'>
1 #TEXT2 (A7) INIT <'AAA BBB'>
1 #TEXT3 (A7) INIT <'AAA-BBB'>
1 #TEXT4 (A7) INIT <'A.B/C,D'>
1 #FIELD1A (A6)
1 #FIELD1B (A6)
1 #FIELD2A (A3)
1 #FIELD2B (A3)
1 #FIELD3A (A3)
1 #FIELD3B (A3)
1 #FIELD4A (A3)
1 #FIELD4B (A3)
1 #FIELD4C (A3)
1 #FIELD4D (A3)
1 #NBT (N1)
1 #DEL (A5)
END-DEFINE
/*****
WRITE NOTITLE 'EXAMPLE A (SOURCE HAS NO BLANKS)'
SEPARATE #TEXT1 INTO #FIELD1A #FIELD1B GIVING NUMBER #NBT
WRITE      / '=' #TEXT1 5X '=' #FIELD1A 4X '=' #FIELD1B 4X '=' #NBT
/*****
WRITE NOTITLE /// 'EXAMPLE B (SOURCE HAS EMBEDDED BLANK)'
SEPARATE #TEXT2 INTO #FIELD2A #FIELD2B GIVING NUMBER #NBT
WRITE      / '=' #TEXT2 4X '=' #FIELD2A 7X '=' #FIELD2B 7X '=' #NBT
/*****
WRITE NOTITLE /// 'EXAMPLE C (USING DELIMITER '-' )'
SEPARATE #TEXT3 INTO #FIELD3A #FIELD3B WITH DELIMITER '-'
WRITE      /      '=' #TEXT3 4X '=' #FIELD3A 7X '=' #FIELD3B
/*****
MOVE ',/' TO #DEL
WRITE NOTITLE /// 'EXAMPLE D USING DELIMITER' '=' #DEL
SEPARATE #TEXT4 INTO #FIELD4A #FIELD4B
                #FIELD4C #FIELD4D WITH DELIMITER #DEL
WRITE      /      '=' #TEXT4 4X '=' #FIELD4A 7X '=' #FIELD4B
            /                19X '=' #FIELD4C 7X '=' #FIELD4D
/*****
END

```

EXAMPLE A (SOURCE HAS NO BLANKS)

#TEXT1: AAABBB #FIELD1A: AAABBB #FIELD1B: #NBT: 1

EXAMPLE B (SOURCE HAS EMBEDDED BLANK)

#TEXT2: AAA BBB #FIELD2A: AAA #FIELD2B: BBB #NBT: 2

EXAMPLE C (USING DELIMITER '-')

#TEXT3: AAA-BBB #FIELD3A: AAA #FIELD3B: BBB

EXAMPLE D USING DELIMITER #DEL: ,/

#TEXT4: A.B/C,D #FIELD4A: A.B #FIELD4B: C
#FIELD4C: D #FIELD4D:

Beispiel 2

```

/* EXAMPLE 'SEPEX2': SEPARATE (USING AN ARRAY)
/*****
DEFINE DATA LOCAL
1 #INPUT-LINE (A60) INIT <'VALUE1,  VALUE2,VALUE3'>
1 #FIELD (A20/1:5)
1 #NUMBER (N2)
END-DEFINE
/*****
SEPARATE #INPUT-LINE LEFT JUSTIFIED INTO #FIELD (1:5)
GIVING NUMBER IN #NUMBER
WRITE NOTITLE #INPUT-LINE //
           #FIELD (1) /
           #FIELD (2) /
           #FIELD (3) /
           #FIELD (4) /
           #FIELD (5) /
           #NUMBER
/*****
END

```

```
VALUE1,  VALUE2,VALUE3
```

```
VALUE1
VALUE2
VALUE3
```

```
3
```

Beispiel 3

```

/* EXAMPLE 'SEPEX3': SEPARATE (REMAINDER, RETAIN)
/*****
DEFINE DATA LOCAL
1 #INPUT-LINE (A60) INIT <'VAL1, VAL2, VAL3,VAL4'>
1 #FIELD (A10/1:4)
1 #REM (A30)
END-DEFINE
WRITE TITLE LEFT 'INP:' #INPUT-LINE /
      '#FIELD (1)' 13T '#FIELD (2)' 25T '#FIELD (3)'
      37T '#FIELD (4)' 49T 'REMAINDER'
/      '_____' 13T '_____' 25T '_____'
      37T '_____' 49T '_____'
/*****
SEPARATE #INPUT-LINE INTO #FIELD (1:2)
      REMAINDER #REM WITH DELIMITERS ', '
WRITE #FIELD(1) 13T #FIELD(2) 25T #FIELD(3) 37T #FIELD(4) 49T #REM
/*****
RESET #FIELD(*) #REM
SEPARATE #INPUT-LINE INTO #FIELD (1:2)
      IGNORE WITH DELIMITERS ', '
WRITE #FIELD(1) 13T #FIELD(2) 25T #FIELD(3) 37T #FIELD(4) 49T #REM
/*****
RESET #FIELD(*) #REM
SEPARATE #INPUT-LINE INTO #FIELD (1:4) IGNORE
      WITH RETAINED DELIMITERS ', '
WRITE #FIELD(1) 13T #FIELD(2) 25T #FIELD(3) 37T #FIELD(4) 49T #REM
/*****
RESET #FIELD(*) #REM
SEPARATE SUBSTRING(#INPUT-LINE,1,50) INTO #FIELD (1:4)
      IGNORE WITH DELIMITERS ', '
WRITE #FIELD(1) 13T #FIELD(2) 25T #FIELD(3) 37T #FIELD(4) 49T #REM
/*****
END

```

INP: VAL1,	VAL2, VAL3,VAL4			
#FIELD (1)	#FIELD (2)	#FIELD (3)	#FIELD (4)	REMAINDER
VAL1	VAL2			VAL3, VAL4
VAL1	VAL2			
VAL1	,	VAL2	,	
VAL1	VAL2	VAL3	VAL4	

SET CONTROL

SET CONTROL *operand1*

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	ja	nein

Funktion

Mit dem Statement SET CONTROL können Sie ein Terminalkommando von einem Programm aus ausführen.

Das Terminalkommando (*operand1*) wird ohne das Kontrollzeichen angegeben und kann als Textkonstante oder als Inhalt einer alphanumerischen Variablen angegeben werden.

Weitere Informationen zu Terminalkommandos finden Sie im *Natural Referenzhandbuch*.

Beispiel 1

```
...
SET CONTROL 'L'
...
```

Schaltet die automatische Umsetzung von Klein- in Großbuchstaben aus (entspricht dem Terminalkommando %L).

Beispiel 2

```
...
SET CONTROL 'HDEST'
...
```

Erzeugt bei der logischen Destination "DEST" eine Hardcopy-Ausgabe (entspricht dem Terminalkommando %HDEST).

SET GLOBALS

Anmerkung:

Dieses Statement ist nur im Reporting Mode erlaubt.

SET GLOBALS *parameter...*

Anstelle dieses Statements können Sie das Systemkommando GLOBALS verwenden.

Funktion

Mit dem Statement SET GLOBALS können Sie Werte für Session-Parameter setzen.

Die Auswertung der Parameter erfolgt je nach Parameter entweder bei der Kompilierung oder bei der Ausführung des Programms, das das SET GLOBALS-Statement enthält.

Die mit SET GLOBALS gesetzten Parameterwerte gelten für die ganze Natural-Session, sofern sie nicht durch ein weiteres SET GLOBALS-Statement (bzw. GLOBALS-Systemkommando) geändert werden.

Ausnahme: Auf Großrechnern gilt ein SET GLOBALS-Statement in einem Unterprogramm (d.h. einer Subroutine, einem Subprogramm oder einem mit FETCH RETURN aufgerufenen Programm) nur solange, bis die Kontrolle von dem Unterprogramm wieder an das aufrufende Objekt übergeben wird; dann gelten wieder die für das aufrufende Objekt gesetzten Parameterwerte.

Parameter

Wenn Sie mehrere Parameter angeben, müssen Sie diese durch ein oder mehrere Leerzeichen voneinander trennen. Die Reihenfolge der Parameter ist beliebig.

Informationen über die einzelnen Parameter finden Sie im Kapitel **Session-Parameter** des *Natural Referenzhandbuchs*.

Beispiel

```
...  
SET GLOBALS LS=74 LT=5000  
...
```

SET KEY

Syntax 1 (für alle Tasten)

$$\text{SET KEY} \left\{ \begin{array}{l} \text{ALL} \\ \text{ON} \\ \text{OFF} \\ \text{COMMAND} \left\{ \begin{array}{l} \text{ON} \\ \text{OFF} \end{array} \right\} \\ \text{NAMED OFF} \end{array} \right\}$$

Syntax 2 (für einzelne Tasten)

$$\text{SET KEY} \left\{ \left\{ \begin{array}{l} \text{PA}_n \\ \text{PF}_n \\ \text{CLR} \\ \text{DYNAMIC } \textit{operand1} \end{array} \right\} \left[= \left\{ \begin{array}{l} \text{ON} \\ \text{OFF} \\ \text{DISABLED} \\ \text{COMMAND} \left\{ \begin{array}{l} \text{ON} \\ \text{OFF} \end{array} \right\} \end{array} \right\} \right] \right\} \dots$$

Syntax 3 (für einzelne Tasten)

$$\text{SET KEY} \left\{ \left\{ \begin{array}{l} \text{PA}_n \\ \text{PF}_n \\ \text{CLR} \\ \text{DYNAMIC } \textit{operand1} \end{array} \right\} \left[= \left\{ \begin{array}{l} \text{PGM} \\ \textit{operand2} \\ \text{HELP} \\ \text{DATA } \textit{operand3} \end{array} \right\} \left[\text{NAMED} \left\{ \begin{array}{l} \textit{operand4} \\ \text{OFF} \end{array} \right\} \right] \right] \right\} \left[\text{ENTR NAMED} \left\{ \begin{array}{l} \textit{operand4} \\ \text{OFF} \end{array} \right\} \right] \dots$$

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S	A	ja	nein
Operand2	C S	A	ja	nein
Operand3	C S	A	ja	nein
Operand4	C S	A	ja	nein

Funktion

Das Statement SET KEY dient dazu, den Funktionstasten von Videoterminals Funktionen zuzuweisen, und zwar den PA-Tasten, den PF-Tasten sowie der CLEAR- bzw. LÖSCH-Taste.

Wird ein SET KEY-Statement ausgeführt, erhält Natural während der Programmausführung die Kontrolle über diese Tasten, und zwar unter Verwendung der Werte, die den Tasten zugewiesen sind.

Über die Natural-Systemvariable *PF-KEY kann ermittelt werden, welche Taste zuletzt gedrückt wurde.

Anmerkungen:

*Wird eine Taste gedrückt, der keine Funktion zugewiesen ist, wird der Benutzer entweder aufgefordert, eine andere Taste zu drücken, oder der Wert "ENTR" wird in die Natural-Systemvariable *PF-KEY gestellt, d.h. Natural reagiert, als ob die ENTER- bzw. FREIG-Taste gedrückt worden wäre (je nachdem, wie Ihr Natural-Administrator den Profilparameter IKEY gesetzt hat).*

Auf Großrechnern hängt die Verarbeitung von PA- und PF-Tasten auch davon ab, wie Ihr Natural-Administrator den Natural-Profilparameter KEY gesetzt hat.

Tasten programm-sensitiv machen

Wenn eine Taste programm-sensitiv gemacht ist, kann Sie vom gerade aktiven Programm abgefragt werden. Das Drücken einer programm-sensitiven Taste hat die gleiche Wirkung wie das Drücken der FREIG-Taste. Alle auf dem Bildschirm eingegebenen Daten werden an das Programm übergeben.

Anmerkung:

Beim Drücken einer programm-sensitiven PA- oder CLEAR-Taste werden keine Daten vom Bildschirm an das Programm übergeben.

Die Programm-Sensitivität ist nur während der Ausführung des aktuellen Programms wirksam. Vgl. Abschnitt **SET KEY-Statements auf verschiedenen Programmebenen** (Seite 604).

Beispiele:**SET KEY ALL**

Dieses Statement bewirkt, daß alle Tasten programm-sensitiv gemacht werden. Alle Funktionszuweisungen zu Tasten werden damit überschrieben.

SET KEY PF2
SET KEY PF2=PGM

Jedes dieser beiden Statements bewirkt, daß PF2 programm-sensitiv gemacht wird.

SET KEY OFF

Dieses Statement deaktiviert alle Funktionstastenbelegungen. (Auf Großrechnern unter Com-plete wird die Kontrolle über die Funktionstasten wieder an den TP-Monitor übergeben.)

SET KEY ON

Dieses Statement reaktiviert die Funktionen aller Tasten, denen eine Funktion zugewiesen war, und macht alle Tasten, die vor der Deaktivierung programm-sensitiv waren, wieder programm-sensitiv.

SET KEY PF2=OFF

Dieses Statement deaktiviert PF2. (Auf Großrechnern unter Com-plete wird die Kontrolle über die Taste wieder an den TP-Monitor übergeben.)

SET KEY PF2=ON

Dieses Statement reaktiviert die Funktion, die PF2 zugewiesen wurde, bevor die Taste deaktiviert oder programm-sensitiv gemacht wurde. War PF2 keine Funktion zugewiesen, wird die Taste wieder programm-sensitiv gemacht.

Kommandos/Programme zuweisen

Sie können einer Taste ein Kommando oder einen Programmnamen zuweisen. Wenn die Taste gedrückt wird, wird das aktuelle Programm unterbrochen und das der Taste zugewiesene Kommando/Programm über den Natural-Stack aufgerufen. Wenn Sie einer Taste ein Kommando/Programm zuweisen, können Sie auch Parameter an das Kommando/Programm übergeben (siehe drittes Beispiel unten).

Sie können einer Taste auch ein Terminalkommando zuweisen. Wenn die Taste gedrückt wird, wird das ihr zugewiesene Terminalkommando ausgeführt.

Wenn *operand2* als Konstante angegeben wird, muß sie in Apostrophen stehen.

Beispiele:

Das Kommando "SAVE" wird PF4 zugewiesen:

```
SET KEY PF4 = 'SAVE'
```

Der in der Variablen #XYZ enthaltene Wert wird PF4 zugewiesen:

```
SET KEY PF4 = #XYX
```

Das Kommando "LIST", einschließlich der LIST-Parameter "MAP" und "*" wird PF6 zugewiesen:

```
SET KEY PF6 = 'LIST MAP *'
```

Das Terminalkommando "%%" wird PF2 zugewiesen:

```
SET KEY PF2='%%'
```

Die Zuweisungen sind solange wirksam, bis sie mit einem anderen SET KEY-Statement überschrieben werden, der Benutzer in eine andere Anwendung wechselt oder die Natural-Session beendet wird. Vgl. Abschnitt **SET KEY-Statements auf verschiedenen Programmebenen** (Seite 604).

Anmerkung:

Bevor ein über eine Taste aufgerufenes Programm ausgeführt wird, führt Natural intern ein BACKOUT TRANSACTION-Statement aus.

Eingabedaten zuweisen (DATA)

Sie können einer Taste eine Datenkette (*operand3*) zuweisen. Wenn die Taste gedrückt wird, werden die Daten in das Eingabefeld übertragen, in dem der Cursor gerade steht, und werden dann an das ausführende Programm übergeben (als ob FREIG gedrückt worden wäre).

Wenn *operand3* als Konstante angegeben wird, muß sie in Apostrophen stehen.

Beispiel:

```
SET KEY PF12=DATA 'YES'
```

Für eine DATA-Zuweisung gilt dasselbe wie für eine Kommando-Zuweisung: sie ist solange wirksam, bis sie mit einem anderen SET KEY-Statement überschrieben wird, der Benutzer in eine andere Anwendung wechselt oder die Natural-Session beendet wird. Vgl. Abschnitt **SET KEY-Statements auf verschiedenen Programmebenen** (Seite 604).

COMMAND OFF/ON

Mit COMMAND OFF können Sie eine Funktion (Kommando, Programm oder Daten), die einer Taste zugewiesen ist, vorübergehend außer Kraft setzen. Wenn die Taste vor der Zuweisung der Funktion programm-sensitiv war, macht COMMAND OFF sie wieder programm-sensitiv.

Mit einem anschließenden COMMAND ON können Sie die zugewiesene Funktion später wieder aktivieren.

Beispiele:

SET KEY PF4=COMMAND OFF

Die PF4 zugewiesene Funktion wird vorübergehend deaktiviert; war PF4 vor der Zuweisung der Funktion programm-sensitiv, wird die Taste jetzt wieder programm-sensitiv.

SET KEY PF4=COMMAND ON

Die PF4 zugewiesene Funktion wird wieder reaktiviert.

SET KEY COMMAND OFF

Die allen Tasten zugewiesenen Funktionen werden vorübergehend deaktiviert; waren Tasten vor der Zuweisung der jeweiligen Funktion programm-sensitiv, werden sie jetzt wieder programm-sensitiv.

SET KEY COMMAND ON

Die allen Tasten zugewiesenen Funktionen werden wieder reaktiviert.

Mit SET KEY PFnn=' ' können Sie die einer Taste zugewiesene Funktion löschen und gleichzeitig die Programm-Sensitivität der Taste deaktivieren.

HELP zuweisen

Sie können einer Taste “HELP” zuweisen. Wenn die Taste gedrückt wird, wird die Helproutine aufgerufen, die dem Feld, in dem der Cursor gerade steht, zugewiesen ist.

Der Effekt ist derselbe wie beim Aufrufen von Hilfe durch Eingabe des Hilfe-Zeichens in das Feld. (Das Hilfe-Zeichen — standardmäßig “?” — wird vom Natural-Administrator mit dem Natural-Profilparameter HI bestimmt.)

Beispiel:

```
SET KEY PF1=HELP
```

Für die HELP-Zuweisung gilt dasselbe wie für Programm-Sensitivität: sie gilt nur für die Ausführung des aktuellen Programms. Vgl. Abschnitt **SET KEY-Statements auf verschiedenen Programmebenen** (Seite 604).

DYNAMIC

Anstatt im SET KEY-Statement eine bestimmte Taste anzugeben, können Sie die Option DYNAMIC mit Angabe einer Variablen (*operand1*) verwenden und dieser Variablen im Programm einen Wert (PFn, PAn, CLR) zuweisen. Dadurch haben Sie die Möglichkeit, eine Funktion anzugeben und es von der Programmlogik abhängig zu machen, welcher Taste diese Funktion zugewiesen wird.

Beispiel:

```
...
IF ...
    MOVE 'PF4' TO #KEY
ELSE
    MOVE 'PF7' TO #KEY
END-IF
...
SET KEY DYNAMIC #KEY = 'SAVE'
...
```

DISABLED

Elemente graphischer Benutzeroberflächen (GUI) wie z.B. Push-Buttons, Menüs und Bitmaps sind als PF-Tasten implementiert. Mit der Option DISABLED können Sie das einer PF-Taste zugewiesene GUI-Element deaktivieren. Das betreffende GUI-Element wird dann grau dargestellt und kann nicht ausgewählt werden.

Mit SET KEY PF nm =ON können Sie SET KEY PF nm =DISABLED wieder rückgängig machen.

Beispiel:

SET KEY PF10=DISABLED

Deaktiviert das PF10 zugewiesene GUI-Element.

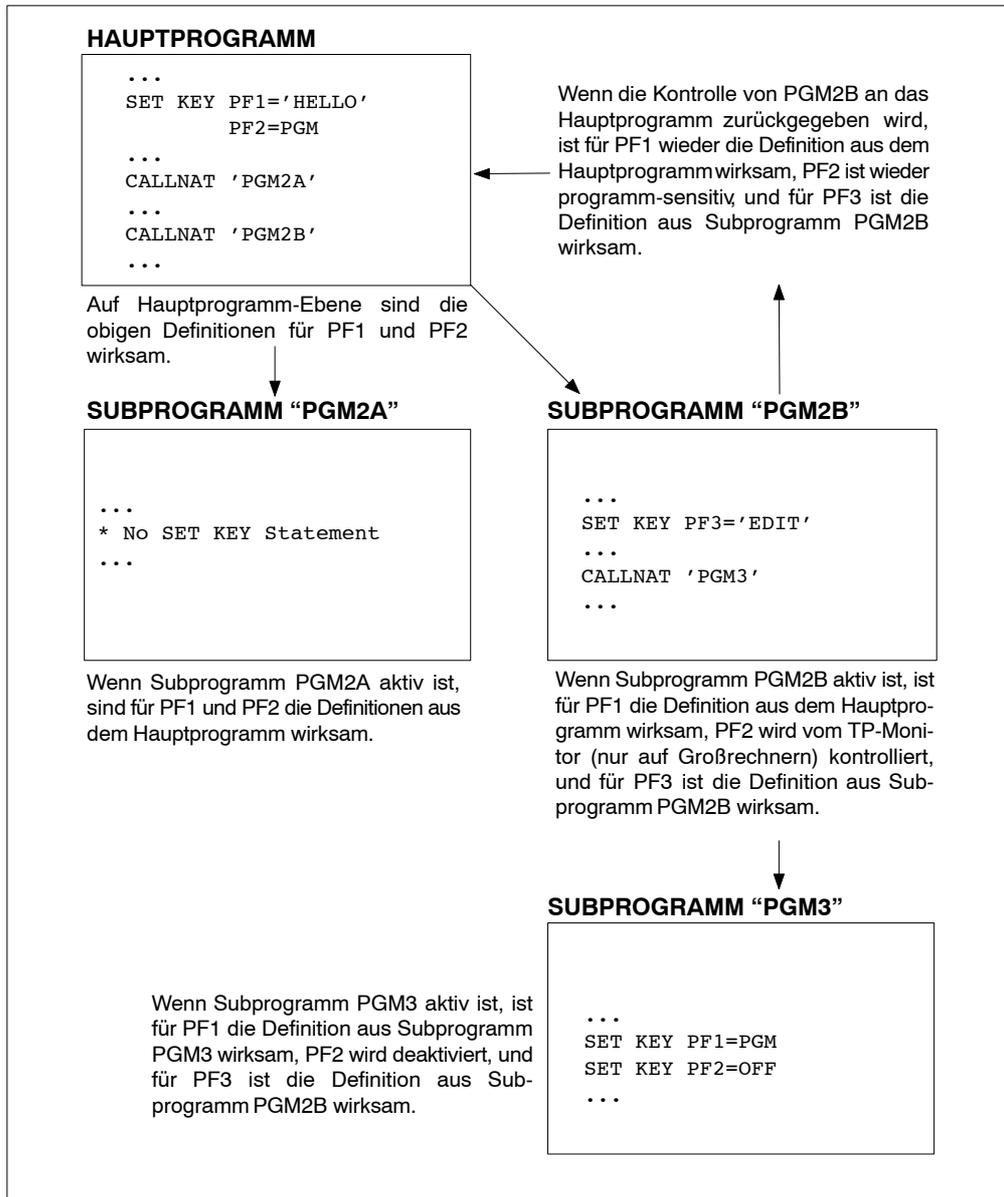
Die DISABLED-Option kann nur in Processing Rules verwendet werden.

SET KEY-Statements auf verschiedenen Programmebenen

Wenn eine Anwendung SET KEY-Statements auf verschiedenen Ebenen enthält, gilt folgendes:

- Wenn Tasten programm-sensitiv gemacht werden, gilt die Programm-Sensitivität auch für alle (aufgerufenen) Programme (bzw. Unterprogramme) auf untergeordneten Ebenen, es sei denn, diese Programme enthalten eigene SET KEY-Statements. Wenn die Kontrolle an ein übergeordnetes Programm zurückgegeben wird, gelten wieder die auf dieser übergeordneten Ebene gemachten SET KEY-Zuweisungen.
- Für Tasten, die als HELP-Tasten definiert sind, gilt das gleiche wie für programm-sensitive Tasten.
- Wenn einer Taste eine Funktion (Programm, Kommando, Terminalkommando oder Daten) zugewiesen wird, gilt diese Zuweisung für alle über- und untergeordneten Ebenen — ganz gleich, auf welcher Ebene die Zuweisung erfolgt —, und zwar solange, bis der Taste eine andere Funktion zugewiesen wird oder sie programm-sensitiv gemacht wird, oder bis der Benutzer in eine andere Anwendung wechselt oder die Natural-Session beendet wird.

Beispiel für SET KEY-Statements auf verschiedenen Programmebenen:



Namen zuweisen

Mit der NAMED-Klausel können Sie einer Taste einen Namen (*operand4*) zuweisen. Dieser Name wird dann in der PF-Tastenleiste auf dem Bildschirm angezeigt, was es dem Benutzer ermöglicht, die den Tasten zugewiesenen Funktionen zu identifizieren:

```

      ?  Help
      .  Exit
-----
Code ... ?  Library ... * _____
            Object ... * _____
            DBID .....: 0__  FILENR ...: 0__

Command ===>
Enter-PF1-PF2-PF3-PF4-PF5-PF6-PF7-PF8-PF9-PF10-PF11-PF12-
      Help      Exit  Last      Flip                          Canc

```

Die Anzeige der PF-Tastenleiste wird mit dem Session-Parameter KD (siehe *Natural Referenzhandbuch*) aktiviert. Die Form der Anzeige der PF-Tastenleiste können Sie mit dem Terminalkommando %Y (siehe *Natural Referenzhandbuch*) beeinflussen.

Der Name, den Sie einer Taste zuweisen, darf bis zu 10 Stellen lang sein. Im normalen tabellarischen PF-Tastenleistenformat (%YN) werden jeweils nur die ersten 5 Stellen angezeigt.

Wenn Sie *operand4* als Konstante angeben, muß diese in Apostrophen stehen (siehe Beispiele).

Sie können einer Taste keinen Namen geben, ohne ihr eine Funktion zuzuweisen oder sie programm-sensitiv zu machen. Der ENTER-Taste können Sie allerdings nur einen Namen zuweisen, aber keine Funktion.

Mit NAMED OFF löschen Sie den Namen einer programm-sensitiven Taste.

Beispiele:**SET KEY ENTR NAMED 'EXEC'**

Der ENTER-Taste wird der Name "EXEC" zugewiesen.

SET KEY PF3 NAMED 'EXIT'

PF3 wird programm-sensitiv gemacht und erhält den Namen "EXIT".

SET KEY PF3 NAMED OFF

PF3 wird programm-sensitiv gemacht, und der PF3 zugewiesene Name wird gelöscht.

SET KEY NAMED OFF

Alle programm-sensitiven Tasten zugewiesenen Namen werden gelöscht.

SET KEY PF4='AP1' NAMED 'APPL1'

PF4 werden das Programm "AP1" und der Name "APPL1" zugewiesen.

Wenn Sie normales tabellarisches PF-Tastenleistenformat (%YN) verwenden, gilt folgendes:

- Wenn Sie einer Taste ein Kommando/Programm zuweisen und die NAMED-Klausel weglassen, wird der Name dieses Kommandos/Programms in der PF-Tastenleiste angezeigt; ist der Name länger als 5 Stellen, wird stattdessen "CMND" angezeigt.
- Wenn Sie einer Taste Eingabedaten zuweisen und die NAMED-Klausel weglassen, wird als Name "DATA" in der PF-Tastenleiste angezeigt.

Wenn Sie sequentielles PF-Tastenleistenformat (%YS oder %YP) verwenden, werden nur die Tasten in der PF-Tastenleiste angezeigt, denen sie Namen zugewiesen haben; d.h. wenn Sie einer Taste ein Kommando/Programm/Daten zuweisen und die NAMED-Klausel weglassen, erscheint die Taste nicht in der PF-Tastenleiste.

Beispiel

```

/* EXAMPLE 'SKYEX1': SET KEY
/*****
DEFINE DATA LOCAL
1 #PF4 (A56)
1 #FCT (A8)
END-DEFINE
/*****
MOVE 'LIST FILES' TO #PF4
/*****
SET KEY PF1 PF2
SET KEY PF3 = 'MENU'
      PF4 = #PF4
      PF5 = 'LIST FILE EMPLOYEES'
/*****
INPUT 10X 'THE FOLLOWING FUNCTION KEYS ARE AVAILABLE:' //
      10X 'PF1: EMPLOYEES UPDATE PROGRAM' /
      10X 'PF2: EMPLOYEES READ PROGRAM' /
      10X 'PF3: RETURN TO MENU          ' /
      10X 'PF4: LIST FILES              ' /
      10X 'PF5: LIST FILE EMPLOYEES    ' ///
      10X '      OR YOU MAY ENTER A PROGRAM TO BE EXECUTED:' #FCT
/*****
IF #FCT NE ' '
  FETCH #FCT
END-IF
IF *PF-KEY = 'PF1'
  FETCH 'UPDPERS'
END-IF
IF *PF-KEY = 'PF2'
  FETCH 'READPERS'
END-IF
/*****
END

```

THE FOLLOWING FUNCTION KEYS ARE AVAILABLE:

PF1: EMPLOYEES UPDATE PROGRAM
 PF2: EMPLOYEES READ PROGRAM
 PF3: RETURN TO MENU
 PF4: LIST FILES
 PF5: LIST FILE EMPLOYEES

OR YOU MAY ENTER A PROGRAM TO BE EXECUTED:

SET TIME

```
{ SETTIME }
{ SET TIME }
```

Funktion

Das Statement SETTIME wird in Verbindung mit der Natural-Systemvariablen *TIMD verwendet und dient dazu, die für die Ausführung eines bestimmten Programmteils benötigte Zeit zu messen.

Das SETTIME-Statement wird an einer bestimmten Stelle im Programm platziert, und die Systemvariable *TIMD gibt dann an, wieviel Zeit seit der Ausführung des SETTIME-Statements verstrichen ist.

Die Systemvariable *TIMD muß das SETTIME-Statement ausdrücklich referenzieren, entweder durch Angabe der Sourcecode-Zeilenummer oder mittels eines Statement-Labels.

Beispiel

```
/* EXAMPLE 'STIEX1': SETTIME
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
END-DEFINE
/*****
ST. SETTIME
WRITE 10X 'START TIME:' *TIME
READ (100) EMPLOY-VIEW BY NAME
END-READ
WRITE NOTITLE 10X 'END TIME: ' *TIME
WRITE          10X 'ELAPSED TIME TO READ 100 RECORDS'
                '(HH:SS:SS.T) : ' *TIMD (ST.) (EM=99:99:99'.9)
/*****
END
```

```
START TIME: 16:34:17.3
END TIME:   16:34:24.0
ELAPSED TIME TO READ 100 RECORDS (HH:II:SS.T) : 00:00:06.7
```

SET WINDOW

```
SET WINDOW { 'window-name'
             OFF }
```

Verwandte Statements

DEFINE WINDOW, INPUT WINDOW='window-name'.

Funktion

Das Statement SET WINDOW dient dazu, ein Bildschirmfenster ("Window") zu aktivieren und zu deaktivieren.

Mit SET WINDOW 'window-name' aktivieren Sie das angegebene Fenster; d.h. alle nachfolgenden Statements beziehen sich auf dieses Fenster, bis es entweder deaktiviert oder ein anderes Fenster aktiviert wird. Das angegebene Fenster muß in einem DEFINE WINDOW-Statement definiert worden sein.

Mit SET WINDOW OFF deaktivieren Sie das gerade aktive Fenster.

Jedes SET WINDOW 'window-name'- oder INPUT WINDOW='window-name'-Statement deaktiviert das gerade aktive Fenster und aktiviert das im Statement angegebene Fenster. Dies bedeutet, daß jeweils nur ein Fenster zur Zeit aktiv sein kann.

Anmerkung:

Wenn Sie mit SET WINDOW ein Fenster aktivieren, das mit SIZE AUTO definiert ist, bestimmen die Daten, die auf dem Schirm sind, bevor das Fenster aktiviert wird, die Größe des Fensters.

Beispiel

Siehe DEFINE WINDOW-Statement.

SKIP

SKIP [(*rep*)] *operand1* [LINES]

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	N P I	ja	nein

Verwandte Statements

DISPLAY, PRINT, WRITE.

Funktion

Das Statement SKIP dient dazu, in einem Ausgabe-Report eine oder mehrere Leerzeilen zu generieren.

Report-Spezifikation (*rep*)

Mit der Notation “(*rep*)” kann ein bestimmter anderer Report angegeben werden, auf den sich das Statement beziehen soll. Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden.

Falls nichts anderes angegeben wird, bezieht sich das SKIP-Statement auf den ersten ausgegebenen Report (Report 0).

Anzahl der Leerzeilen (*operand1*)

Operand1 ist die Anzahl der zu generierenden Leerzeilen (1 – 250). Die Anzahl kann als numerische Konstante oder als Inhalt einer numerischen Variablen angegeben werden.

Ist *operand1* größer als die für den Report definierte Seitenlänge, so löst das SKIP-Statement eine “Newpage”-Bedingung aus.

Sonstige Hinweise

Wenn bei der Ausführung eines SKIP-Statements die in den Report einzufügenden Leerzeilen nicht mehr auf die aktuelle Ausgabeseite passen, werden die überschüssigen Leerzeilen ignoriert (außer in einem AT TOP OF PAGE-Statement).

Ein SKIP-Statement wird nur ausgeführt, falls vorher auf der Seite bereits etwas ausgegeben wurde (eine über ein AT TOP OF PAGE-Statement erzeugte Ausgabe wird hierbei nicht berücksichtigt).

Beispiel

```

/* EXAMPLE 'SKPEX1': SKIP
/*****
LIMIT 7
READ EMPLOYEES BY CITY STARTING FROM 'W'
  AT BREAK OF CITY
  SKIP 2
  DISPLAY NOTITLE CITY (IS=ON) COUNTRY (IS=ON) NAME
/*****
END

```

CITY	COUNTRY	NAME
WASHINGTON	USA	REINSTEDT PERRY
WEITERSTADT	D	BUNGERT UNGER DECKER
WEST BRIDGFORD	UK	ENTWHISTLE
WEST MIFFLIN	USA	WATSON

SORT

Structured-Mode-Syntax

```

END-ALL
[AND]
SORT [ THEM
         RECORDS ] [BY] { operand1 [ ASCENDING
         DESCENDING ] } ...10
    USING-clause
    [GIVE-clause]
    statement...
END-SORT

```

- * Wenn ein Statement-Label angegeben wird, muß es vor dem Schlüsselwort SORT, aber nach END-ALL (und AND) stehen.

Reporting-Mode-Syntax

```

SORT [ THEM
         RECORDS ] [BY] { operand1 [ ASCENDING
         DESCENDING ] } ...10
    [USING-clause]
    [GIVE-clause]
    statement...

```

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S	A N P I F B D T	nein	nein

Verwandtes Statement

FIND mit SORTED BY-Option.

Funktion

Das Statement SORT dient dazu, eine Sortieroperation durchzuführen und die Datensätze aus allen Verarbeitungsschleifen, die zum Zeitpunkt der SORT-Ausführung aktiv sind, zu sortieren.

Für die Sortieroperation wird Natural's internes Sortierprogramm verwendet. Auf Großrechnern kann stattdessen auch ein anderes, externes Sortierprogramm verwendet werden. Welches Sortierprogramm verwendet wird, legt der Natural-Administrator im Makro NTSORT des Natural-Parametermoduls fest (vgl. *Natural Operations for Mainframes Documentation*).

Bei der Verwendung eines externen Sortierprogramms ist zusätzliche JCL erforderlich; bitte wenden Sie sich hierzu an Ihren Natural-Administrator.

Anmerkung:

Unter OpenVMS und UNIX werden die zu sortierenden Datensätze in dem Verzeichnis zwischengespeichert, das in der Konfigurationsdatei "NATURAL.INI" unter "TMP_PATH" angegeben ist.

Einschränkungen

Das SORT-Statement muß im selben Objekt stehen wie die Verarbeitungsschleifen, deren Datensätze es sortiert.

Geschachtelte SORT-Statements sind nicht erlaubt.

Auf Großrechnern darf die Gesamtlänge eines zu sortierenden Datensatzes 10240 Bytes nicht überschreiten.

Verarbeitungsschleifen

Im *Reporting Mode* bewirkt das SORT-Statement, daß alle noch aktiven Verarbeitungsschleifen beendet und eine neue Verarbeitungsschleife initiiert wird.

Im *Structured Mode* müssen Sie vor dem SORT-Statement END-ALL angeben; damit werden alle noch aktiven Verarbeitungsschleifen beendet. Das SORT-Statement initiiert seinerseits eine neue Verarbeitungsschleife, welche mit END-SORT geschlossen werden muß.

Sortierkriterien (*operand1*)

Operand1 sind die Felder/Variablen, die als Sortierkriterium dienen. 1 bis 10 Felder dürfen angegeben werden. Hierbei kann es sich um Datenbankfelder (Deskriptoren oder Nicht-Deskriptoren) und/oder Benutzervariablen handeln. Multiple Felder oder Felder aus einer Periodengruppe können ebenfalls verwendet werden; eine Feldgruppe oder ein Array können nicht verwendet werden.

Wird nichts anderes angegeben, so gilt ASCENDING, d.h. die Werte werden in aufsteigender Reihenfolge sortiert. Möchten Sie die Werte in absteigender Reihenfolge sortiert haben, geben Sie das Schlüsselwort DESCENDING an. ASCENDING/DESCENDING kann für jedes Sortierfeld getrennt angegeben werden.

USING-clause

```
{ USING operand2...
  USING KEYS }
```

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand2	S A	A N P I F B D T L C	nein	nein

In der USING-Klausel geben Sie die Felder an, die in den Sortier-Zwischenspeicher geschrieben werden sollen. Die USING-Klausel ist im *Structured Mode* unbedingt erforderlich, im *Reporting Mode* nicht; allerdings wird dringend empfohlen, sie auch im *Reporting Mode* zu verwenden, um Speicherplatz zu sparen.

Wenn Sie USING KEYS angeben, werden nur die als *operand1* angegebenen Sortierfelder in den Sortier-Zwischenspeicher geschrieben.

Mit USING *operand2* können Sie weitere Felder angeben, die — zusätzlich zu den (als *operand1* angegebenen) Sortierfeldern — in den Sortier-Zwischenspeicher geschrieben werden sollen.

Verwenden Sie im *Reporting Mode* keine USING-Klausel, so werden alle Datenbankfelder aus vor dem SORT-Statement initiierten Verarbeitungsschleifen sowie alle vor dem SORT-Statement definierten Benutzervariablen in den Sortier-Zwischenspeicher geschrieben.

Wird nach Ausführung des SORT-Statements ein Feld referenziert, das nicht in den Sortier-Zwischenspeicher geschrieben wurde, so ist der Wert des Feldes der, den es vor der SORT-Operation hatte.

GIVE-clause

GIVE	$\left\{ \begin{array}{l} \text{MAX} \\ \text{MIN} \\ \text{NMIN} \\ \text{COUNT} \\ \text{NCOUNT} \\ \text{OLD} \\ \text{AVER} \\ \text{NAVER} \\ \text{SUM} \\ \text{TOTAL} \\ \dots \end{array} \right.$	$[\text{OF}] \left\{ \begin{array}{l} (\text{operand3} \dots) \\ \text{operand3} \dots \end{array} \right.$	$[(\text{NL}=\text{nn})]$	\dots
------	---	---	---------------------------	---------

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand3	S A	*	ja	nein

* je nach Funktion

Die GIVE-Klausel dient dazu, Natural-Systemfunktionen (MAX, MIN, usw.) anzugeben, die in der ersten Phase des Sortiervorgangs ausgewertet werden und dann in der dritten Phase referenziert werden können (siehe Abschnitt **Phasen der SORT-Verarbeitung**). Wird nach dem SORT-Statement eine Systemfunktion referenziert, muß ihrem Namen ein Stern vorangestellt werden; Beispiel: *AVER(SALARY).

Näheres zu den einzelnen Systemfunktionen finden Sie im *Natural Referenzhandbuch*.

(NL=nn)

Diese Option kann dazu verwendet werden, einen arithmetischen Überlauf bei der Auswertung von Systemfunktionen zu vermeiden; sie ist unter **Arithmetischer Überlauf bei AVER, NAVER, SUM oder TOTAL** im Kapitel **Systemfunktionen** des *Natural Referenzhandbuchs* beschrieben.

Diese Option gilt nur für AVER, NAVER, SUM und TOTAL; für alle anderen Systemfunktionen wird sie ignoriert.

Phasen der SORT-Verarbeitung

Ein Programm, das ein SORT-Statement enthält, wird in drei Phasen ausgeführt:

1. Phase — Auswählen der zu sortierenden Datensätze

Die Statements vor dem SORT-Statement werden ausgeführt. Die in der USING-Klausel angegebenen Daten werden in den Sortier-Zwischenspeicher geschrieben.

Im *Reporting Mode* dürfen Variablen, die nach dem Sortieren als Akkumulatoren verwendet werden, nicht vor dem SORT-Statement definiert werden. Im *Structured Mode* dürfen sie nicht in der USING-Klausel angegeben werden. In den Sortier-Zwischenspeicher geschriebene Felder können als Akkumulatoren nicht verwendet werden, weil sie in der dritten Phase mit jedem einzelnen Datensatz zurückgeschrieben werden. Folglich hätte die Akkumulationsfunktion nicht das gewünschte Ergebnis, da das Feld bei jedem Datensatz mit dem Wert des jeweiligen Datensatzes überschrieben würde.

Die Anzahl der in den Sortier-Zwischenspeicher geschriebenen Datensätze ergibt sich aus der Anzahl der Verarbeitungsschleifen und der Anzahl der verarbeiteten Datensätze pro Schleife. Jedesmal wenn das SORT-Statement in einer Verarbeitungsschleife ausgeführt wird, wird im internen Sortier-Zwischenspeicher ein Datensatz angelegt.

Bei geschachtelten Schleifen wird ein Datensatz nur in den Sortier-Zwischenspeicher geschrieben, wenn die innere Schleife ausgeführt wird. Sollen im folgenden Beispiel Datensätze in den Sortier-Zwischenspeicher geschrieben werden, auch wenn in der inneren (FIND-)Schleife keine gefunden werden, so muß das FIND-Statement eine IF NO RECORDS FOUND-Klausel enthalten.

```
READ ...
...
  FIND ...
...
END-ALL
SORT ...
  DISPLAY ...
END-SORT
...
```

2. Phase — Sortieren der Datensätze

Die Datensätze werden sortiert.

3. Phase — Weiterverarbeitung der sortierten Datensätze

Die Datensätze aus dem Sortier-Zwischenspeicher werden in der angegebenen Sortierfolge mit den auf das SORT-Statement folgenden Statements weiterverarbeitet.

Werden Datenbankfelder nach dem SORT-Statement referenziert, so muß dies über ein Statement-Label oder durch Angabe der entsprechenden Sourcecode- Zeilennummer erfolgen.

Beispiel

```

/* EXAMPLE 'SRTEX1R': SORT (REPORTING MODE)
/*****
LIMIT 3
FIND EMPLOYEES WITH CITY = 'BOSTON'
OBTAIN SALARY(1:2)
COMPUTE #TOTAL-SALARY (P11) = SALARY (1) + SALARY (2)
ACCEPT IF #TOTAL-SALARY GT 0
/*****
SORT BY PERSONNEL-ID USING #TOTAL-SALARY SALARY(*) CURR-CODE
GIVE AVER(#TOTAL-SALARY)
/*****
AT START OF DATA
DO
WRITE NOTITLE
  '**' (40)
  'AVG CUMULATIVE SALARY:' *AVER (#TOTAL-SALARY) /
  MOVE *AVER (#TOTAL-SALARY) TO #AVG (P11)
DOEND
COMPUTE #AVER-PERCENT (N3.2) = #TOTAL-SALARY / #AVG * 100
ADD #TOTAL-SALARY TO #TOTAL-TOTAL (P11)
DISPLAY NOTITLE PERSONNEL-ID SALARY (1) SALARY (2)
  #TOTAL-SALARY CURR-CODE (1)
  'PERCENT/OF/AVER' #AVER-PERCENT
AT END OF DATA
WRITE / '**' (40) 'TOTAL SALARIES PAID: ' #TOTAL-TOTAL
/*****
END

```

PERSONNEL ID	ANNUAL SALARY	ANNUAL SALARY	#TOTAL-SALARY	CURRENCY CODE	PERCENT OF AVER
***** AVG CUMULATIVE SALARY:					41900
20007000	16000	15200	31200	USD	74.00
20019200	18000	17100	35100	USD	83.00
20020000	30500	28900	59400	USD	141.00
***** TOTAL SALARIES PAID:					125700

Das Beispiel auf der vorherigen Seite wird wie folgt verarbeitet:

Erste Phase:

- Von der EMPLOYEES-Datei werden Datensätze mit CITY = BOSTON gelesen.
- Die ersten beiden Ausprägungen des SALARY-Feldes werden in der Variablen #TOTAL-SALARY zusammenaddiert.
- Es werden nur Datensätze weiterverarbeitet, bei denen der Wert von #TOTAL-SALARY größer als 0 ist.
- Die Sätze werden in den Sortier-Zwischenspeicher geschrieben. Die Datenbank-Arrays SALARY (die ersten beiden Ausprägungen) und CURR-CODE (erste Ausprägung), das Datenbankfeld PERSONNEL-ID und die Benutzervariable #TOTAL-SALARY werden in den Zwischenspeicher geschrieben.
- Der Durchschnittswert von #TOTAL-SALARY wird errechnet.

Zweite Phase:

- Die Datensätze werden sortiert.

Dritte Phase:

- Der sortierte Zwischenspeicher wird gelesen.
- Bei der Ausführung des AT START OF DATA-Blocks wird der Durchschnittswert von #TOTAL-SALARY angezeigt.
- Der Wert von #TOTAL-SALARY wird zu #TOTAL-TOTAL hinzuaddiert; es werden die Felder PERSONNEL-ID, SALARY (1), SALARY (2), #AVER-PERCENT und #TOTAL-SALARY angezeigt.
- Bei der Ausführung des AT END OF DATA-Blocks wird die Variable #TOTAL-TOTAL ausgegeben.

STACK

$\text{STACK [TOP]} \left\{ \begin{array}{l} \text{COMMAND } \textit{operand1} \left[\textit{operand2} [(parameter)] \dots \right] \\ \text{[DATA] [FORMATTED] } \{ \textit{operand2} [(parameter)] \dots \} \end{array} \right\}$

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S A G N	A	ja	nein
Operand2	C S A G N	A N P I F B D T L G	ja	ja

Verwandte Statements

INPUT, RELEASE.

Funktion

Das Statement STACK dient dazu, Daten im Natural-Stack abzulegen. Hierbei kann es sich um folgende Daten handeln:

- den Namen eines Natural-Programms oder -Systemkommandos, das ausgeführt werden soll
- Daten, die bei der Ausführung eines INPUT-Statements als Eingabedaten verwendet werden sollen.

Weitere Informationen zum Stack finden Sie im Kapitel **Weitere Programmieraspekte** des *Natural Leitfadens zur Programmierung*.

TOP

Normalerweise werden die Daten *unten* im Stack abgelegt. Das Schlüsselwort TOP bewirkt, daß die Daten *oben* auf dem Stack abgelegt werden.

Beispiel:

Mit diesem Statement wird der Inhalt der Variablen #FIELD1 oben auf dem Stack abgelegt:

```
STACK TOP #FIELD1
```

DATA

Mit DATA legen Sie Daten im Stack ab, die von einem INPUT-Statement als Eingabedaten verwendet werden.

Delimiterzeichen oder "Input Assign"-Zeichen innerhalb der übergebenen Datenwerte werden als Delimiter interpretiert und entsprechend verarbeitet. Einzelheiten darüber, wie die Daten von einem INPUT-Statement verarbeitet werden, können Sie der Beschreibung des INPUT-Statements entnehmen.

Beispiel:

Mit den folgenden Statements wird der Inhalt der Variablen #FIELD1 und #FIELD2 im Stack abgelegt:

```
MOVE 'ABC' TO #FIELD1  
MOVE 'XYZ' TO #FIELD2  
STACK #FIELD1 #FIELD2
```

Diese Variablen werden als Eingabedaten an das nächste INPUT-Statement im Natural-Programm übergeben, und zwar im Delimiter-Modus:

```
INPUT #FIELD1 #FIELD2
```

Anmerkung:

Wenn operand2 eine Zeitvariable (Format T) ist, wird nur die Zeitkomponente des Variableninhalts im Stack abgelegt, aber nicht die Datumskomponente.

FORMATTED

Das Schlüsselwort `FORMATTED` bewirkt, daß alle Daten Feld für Feld an das nächste `INPUT`-Statement übergeben werden. Schlüsselzuordnungen oder Delimiterzeichen werden nicht als solche interpretiert.

Beispiele:

Mit den folgenden Statements wird “ABC,DEF” in `#FIELD1` übertragen und “XYZ” in `#FIELD2`:

```
MOVE 'ABC,DEF' TO #FIELD1
MOVE 'XYZ'     TO #FIELD2
STACK TOP DATA FORMATTED #FIELD1 #FIELD2
...
INPUT #FIELD1 #FIELD2
```

Angenommen, das Input-Delimiterzeichen ist das Komma (ID=), dann wird mit folgenden Statements — ohne das Schlüsselwort `FORMATTED` — “ABC” in `#FIELD1` übertragen und “DEF” in `#FIELD2`:

```
MOVE 'ABC,DEF' TO #FIELD1
STACK TOP DATA #FIELD1
...
INPUT #FIELD1 #FIELD2
```

COMMAND *operand1*

Um ein Kommando (bzw. einen Programmnamen) im Stack abzulegen, geben Sie das Schlüsselwort COMMAND gefolgt von dem betreffenden Kommando (*operand1*) an. Würde ein Programm normalerweise den Benutzer mit einer Eingabeaufforderung in Form einer NEXT-Zeile konfrontieren, so unterdrückt nun Natural die Anzeige der NEXT-Zeile und führt stattdessen das im Stack abgelegte Kommando aus.

Beispiel:

Mit dem folgenden Statement wird das Kommando RUN oben auf dem Stack abgelegt. Natural führt dieses Kommando aus, wenn normalerweise das nächstemal die NEXT-Zeile ausgegeben würde.

```
STACK TOP COMMAND 'RUN'
```

COMMAND *operand1 operand2...*

Zusammen mit einem Kommando (*operand1*) können Sie auch Daten (*operand2*) im Stack ablegen. Diese Daten werden dann vom nächsten INPUT-Statement nach der Ausführung des Kommandos als Eingabedaten verarbeitet.

Zusammen mit einem Kommando abgelegte Daten werden immer unformatiert abgelegt.

Anmerkung:

Wenn in den abzulegenden Daten leere alphanumerische Felder (d.h. Leerzeichen) enthalten sind, werden diese Leerzeichen als Delimiter zwischen Werten interpretiert und folglich von dem betreffenden INPUT-Statement falsch verarbeitet. Wenn Sie daher leere alphanumerische Felder als Daten zusammen mit einem Kommando im Stack ablegen möchten, müssen Sie hierzu zwei STACK-Statements verwenden: ein "STACK DATA operand2...", um die Daten abzulegen, und ein "STACK COMMAND operand1", um das Kommando abzulegen.

parameter

Wenn *operand2* eine Datumsvariable ist, können Sie den Session-Parameter DF als *parameter* für diese Variable angeben. Der Session-Parameter DF ist im *Natural Referenzhandbuch* beschrieben.

Beispiel

```

/* EXAMPLE 'STKEX1': STACK
/*****
INPUT 'PLEASE SELECT DESIRED FUNCTION:' //
  10X 'LIST FILES (F)' /
  10X 'LIST PROGRAMS (P)' /
  10X 'FUNCTION:' #RSP (A1)
/*****
IF NOT (#RSP = 'F' OR = 'P')
  REINPUT 'PLEASE ENTER A CORRECT FUNCTION'
/*****
IF #RSP = 'F'
  DO STACK TOP COMMAND 'LIST FILES *' STOP DOEND
/*****
IF #RSP = 'P'
  DO STACK TOP COMMAND 'LIST PROGRAMS *' STOP DOEND
/*****
END

```

PLEASE SELECT DESIRED FUNCTION:

LIST FILES (F)
LIST PROGRAMS (P)
FUNCTION: p

```

16:51:17          ***** NATURAL LIST COMMAND *****          03-27-87
USER: TM          LIST                                          LIB: RJNV2RM
C Name           Pgm. Type   SM S/C Vers Level Userid   Time    Date
-----
SISXXX          Program      S S  2.1  0000  RJ      12:06  87:03:13
SKPEX1          Program      R S  2.1  0000  RJ      19:24  87:03:23
SKYEX1          Program      S S  2.1  0000  RJ      19:02  87:03:23
SRTEX1          Program      R S  2.1  0000  RJ      10:11  87:03:13
SRTEX1R         Program      R S  2.1  0000  RJ      09:44  87:03:24
SRTEX1S         Program      S S  2.1  0000  RJ      09:48  87:03:24
SRTEX2          Program      S S  2.1  0000  RJ      10:35  87:03:13
STIEX1          Program      S S  2.1  0000  RJ      19:21  87:03:23
STKEX1          Program      R S  2.1  0000  RJ      09:54  87:03:24
STKEX1R         Program      R S  2.1  0000  RJ      10:46  87:03:13
STOEX1          Program      S S  2.1  0000  RJ      11:28  87:03:13
STOEX1R         Program      R S  2.1  0000  RJ      10:45  87:03:24
STOEX1S         Program      S S  2.1  0000  RJ      10:14  87:03:24
STOEX1T         Program      R S  2.1  0000  RJ      19:35  87:03:24
STOEX1U         Program      R S  2.1  0000  RJ      19:42  87:03:24

```

STOP

STOP

Funktion

Mit dem Statement `STOP` können Sie die Ausführung eines Programmes abbrechen und erhalten dann eine Kommandozeile.

Sie können ein `STOP`-Statement an beliebiger Stelle im Programm verwenden und auch mehrere `STOP`-Statements benutzen.

Mit dem `STOP`-Statement wird die Ausführung des Programms sofort abgebrochen. Befindet sich das `STOP`-Statement in einer Subroutine, so wird vor dem Abbruch noch eine etwaige im Hauptprogramm angegebene Seitenende-Bedingung (`end-of-page`) zur abschließenden Seitenende-Verarbeitung ausgeführt.

Beispiel

```
/* EXAMPLE 'STPEX1': STOP
/*****
INPUT 'PLEASE SELECT DESIRED FUNCTION:' //
  10X 'LIST FILES (F)' /
  10X 'LIST PROGRAMS (P)' /
  10X 'FUNCTION:' #RSP (A1)
/*****
IF #RSP = ' '
  STOP
/*****
IF NOT (#RSP = 'F' OR = 'P')
  REINPUT 'PLEASE ENTER A CORRECT FUNCTION'
/*****
IF #RSP = 'F'
  DO
    STACK TOP COMMAND 'LIST FILES *'
    STOP
  DOEND
/*****
IF #RSP = 'P'
  DO
    STACK TOP COMMAND 'LIST PROGRAMS *'
    STOP
  DOEND
/*****
END
```

STORE

Structured-Mode-Syntax

```

STORE [RECORD] [IN] [FILE] view-name
      [PASSWORD=operand1]
      [CIPHER=operand2]
      [ [USING
        GIVING] NUMBER operand3 ] [(r)]

```

Reporting-Mode-Syntax

```

STORE [RECORD] [IN] [FILE] view-name
      [PASSWORD=operand1]
      [CIPHER=operand2]
      [ [USING
        GIVING] NUMBER operand3 ]
      { [USING] SAME [RECORD][AS][STATEMENT [(r)]]
        [SET WITH] [operand4=operand5] ... }

```

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	ja	nein
Operand2	C S	N	ja	nein
Operand3	S	N P	nein	ja
Operand4	S A	A N P I F B D T L	nein	nein
Operand5	C S A	A N P I F B D T L	ja	nein

Verwandte Statements

UPDATE, DELETE, END TRANSACTION, BACKOUT TRANSACTION.

Funktion

Das Statement STORE dient dazu, auf einer Datenbank einen Datensatz hinzuzufügen.

Anmerkungen für DL/I-Datenbanken:

Mit dem STORE-Statement kann eine Segment-Ausprägung hinzugefügt werden.

Ist der Datensatz mit einem Primärschlüssel definiert, so muß ein Wert für das Primärschlüssel-Feld angegeben werden.

Bei einer GSAM-Datenbank müssen neue Datensätze am Ende der Datenbank hinzugefügt werden (aufgrund von GSAM-Beschränkungen).

Anmerkung für SQL-Datenbanken:

Mit dem STORE-Statement können Sie einer Tabelle eine Reihe hinzufügen. Die PASSWORD-, CIPHER- und GIVING NUMBER-Klauseln sind nicht erlaubt. Das STORE-Statement entspricht dem SQL-Statement INSERT.

Anmerkung für VSAM-Datenbanken:

Ist der Datensatz mit einem Primärschlüssel definiert, so muß ein Wert für das Primärschlüssel-Feld angegeben werden.

view-name

Als *view-name* geben Sie den Namen eines Views an, der entweder in einem DEFINE DATA-Block oder in einer programmexternen Global oder Local Data Area definiert ist.

Im *Reporting Mode* darf — falls kein DEFINE DATA-Statement verwendet wird — *view-name* auch der Name eines DDMS sein.

PASSWORD/CIPHER

Diese Klauseln gelten nur für Adabas- und VSAM-Datenbanken.

Die PASSWORD-Klausel dient dazu, ein Paßwort anzugeben, um Daten auf einer paßwort-geschützten Datei speichern zu können.

Die CIPHER-Klausel dient dazu, einen Cipher-Code (Chiffrierschlüssel) anzugeben, um Daten auf einer chiffrierten Datei in verschlüsselter Form speichern zu können.

Weitere Informationen siehe Statements FIND und PASSW.

USING/GIVING NUMBER

Diese Klausel gilt nur für Adabas- und VSAM-Datenbanken.

Mit dieser Option können Sie für einen zu speichernden Datensatz eine eigene Adabas-ISN (Interne Satznummer) angeben. Ist die angegebene ISN bereits vergeben, wird das Programm abgebrochen und eine entsprechende Fehlermeldung ausgegeben, es sei denn, es ist eine ON ERROR-Verarbeitung definiert.

Anmerkung für VSAM-Datenbanken:

Diese Klausel gilt nur für VSAM-RRDS, wobei die RRN (relative record number) der ISN entspricht.

SET/WITH

Mit dieser Klausel können im *Reporting Mode* die Felder angegeben werden, für die Werte gespeichert werden sollen. Jedes in der Datei definierte Feld, das in der SET-Klausel nicht angegeben wird, erhält in dem neuen Datensatz einen Nullwert.

Diese Klausel ist nicht erlaubt, wenn ein DEFINE DATA-Statement verwendet wird, da sich in diesem Fall das STORE-Statement immer auf den gesamten im DEFINE DATA-Statement definierten View bezieht.

Hinweise für DL/I

Es müssen für das Segment-Folgefild sowie für alle Folgefilder der “Ancestors” Werte angegeben werden.

Es dürfen nur Werte für I/O-sensitive Felder angegeben werden.

Ein Segment mit variabler Länge wird gespeichert mit der Länge, die mindestens benötigt wird, um alle im STORE-Statement angegebenen Felder aufnehmen zu können. Die Segmentlänge unterschreitet jedoch nie die im SEGM-Makro der DBD definierte Mindestlänge.

Wird ein multiples Feld oder eine Periodengruppe mit variabler Länge definiert, so werden am Ende des Segments nur die im STORE-Statement angegebenen Ausprägungen zu dem Segment hinzugeschrieben, welche die Segmentlänge bestimmen.

USING SAME

Diese Klausel bewirkt im *Reporting Mode*, daß dieselben Feldwerte, die mit dem FIND-, GET- oder READ-Statement gelesen wurden, welches von dem STORE-Statement referenziert wird, als Werte des neuen Datensatzes gespeichert werden.

Diese Klausel ist nicht erlaubt, wenn ein DEFINE DATA-Statement verwendet wird, da sich in diesem Fall das STORE-Statement immer auf den gesamten im DEFINE DATA-Statement definierten View bezieht.

Systemvariable *ISN

Die Natural-Systemvariable enthält die Adabas-ISN bzw. VSAM-RBA oder -RRN, die dem neuen Datensatz als Ergebnis der STORE-Operation zugeteilt wurde. Um den Wert von *ISN zu erhalten, müssen Sie das betreffende STORE-Statement referenzieren (mittels Statement-Label oder Sourcecode-Zeilenummer).

Bei VSAM-Dateien gilt *ISN nur für ESDS- und RRDS-Dateien.

Für DL/I- und SQL-Datenbanken ist *ISN nicht verfügbar.

Beispiel

```

* EXAMPLE 'STOEX1': STORE
*****
RESET #BIRTH-D (D)
*
REPEAT
  INPUT 'ENTER A PERSONNEL ID AND NAME (OR ''END'' TO END)' //
        'PERSONNEL-ID : ' #PERSONNEL-ID (A8) //
        'NAME          : ' #NAME          (A20) /
        'FIRST-NAME    : ' #FIRST-NAME    (A15)
*****
*   VALIDATE ENTERED DATA
*
  IF #PERSONNEL-ID = 'END' OR #NAME = 'END' THEN STOP
  IF #NAME = ' ' THEN
    REINPUT WITH TEXT 'ENTER A LAST-NAME' MARK 2 AND SOUND ALARM
  IF #FIRST-NAME = ' ' THEN
    REINPUT WITH TEXT 'ENTER A FIRST-NAME' MARK 3 AND SOUND ALARM
*****
*   ENSURE PERSON IS NOT ALREADY ON FILE
*
  FIND NUMBER EMPLOYEES WITH PERSONNEL-ID = #PERSONNEL-ID
  IF *NUMBER > 0 THEN
    REINPUT 'PERSON WITH SAME PERSONNEL-ID ALREADY EXISTS'
    MARK 1 AND SOUND ALARM
*****
*   GET FURTHER INFORMATION
*
  INPUT
    'ADDITIONAL PERSONNEL DATA' //
    'PERSONNEL-ID          : ' #PERSONNEL-ID (AD=IO) /
    'NAME                  : ' #NAME          (AD=IO) /
    'FIRST-NAME            : ' #FIRST-NAME    (AD=IO) ///
    'MARITAL STATUS        : ' #MAR-STAT (A1) /
    'DATE OF BIRTH (YYYYMMDD) : ' #BIRTH      (A8) /
    'CITY                  : ' #CITY          (A20) //
    'COUNTRY (3 CHARACTERS) : ' #COUNTRY     (A3) //
    'ADD THIS RECORD (Y/N)   : ' #CONF       (A1) (AD=M)
*****
*   ENSURE REQUIRED FIELDS CONTAIN VALID DATA
*
  IF NOT (#MAR-STAT = 'S' OR = 'M' OR = 'D' OR = 'W')
    REINPUT TEXT 'ENTER VALID MARITAL STATUS S=SINGLE ' -
                'M=MARRIED D=DIVORCED W=WIDOWED' MARK 1
  IF NOT (#BIRTH = MASK(YYYYMMDD) AND #BIRTH = MASK(1582-2699))
    REINPUT TEXT 'ENTER CORRECT DATE' MARK 2
  IF #CITY = ' '
    REINPUT TEXT 'ENTER A CITY NAME' MARK 3
  IF #COUNTRY = ' '
    REINPUT TEXT 'ENTER A COUNTRY CODE' MARK 4
  IF NOT (#CONF = 'N' OR = 'Y')
    REINPUT TEXT 'ENTER Y (YES) OR N (NO)' MARK 5
  IF #CONF = 'N'
    ESCAPE TOP
*****
* Fortsetzung nächste Seite

```

```

* Fortsetzung von vorheriger Seite
*****
* ADD THE RECORD
*
MOVE EDITED #BIRTH TO #BIRTH-D (EM=YYYYMMDD)
*
STORE RECORD IN EMPLOYEES
WITH PERSONNEL-ID = #PERSONNEL-ID
NAME = #NAME
FIRST-NAME = #FIRST-NAME
MAR-STAT = #MAR-STAT
BIRTH = #BIRTH-D
CITY = #CITY
COUNTRY = #COUNTRY
END OF TRANSACTION
*
WRITE NOTITLE 'RECORD HAS BEEN ADDED'
RESET INITIAL #CONF
*
LOOP
END

```

ENTER A PERSONNEL ID AND NAME (OR 'END' TO END)

PERSONNEL-ID : 90001100

NAME : JONES

FIRST-NAME : EDWARD

ADDITIONAL PERSONNEL DATA

PERSONNEL-ID : 90001100

NAME : JONES

FIRST-NAME : EDWARD

MARITAL STATUS : m

DATE OF BIRTH (YYYYMMDD) : 690511

CITY : wan chai

COUNTRY (3 CHARACTERS) : hkg

ADD THIS RECORD (Y/N) : y

SUBTRACT

Syntax 1

SUBTRACT [ROUNDED] *operand1*... **FROM** *operand2*

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S A N	N P I F D T	ja	nein
Operand2	S A M	N P I F D T	ja	nein

Syntax 2

SUBTRACT [ROUNDED] *operand1*... **FROM** *operand2* **GIVING** *operand3*

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S A N	N P I F D T	ja	nein
Operand2	C S A N	N P I F D T	ja	nein
Operand3	S A M	A N P I F B D T	ja	ja

Verwandtes Statement

COMPUTE.

Funktion

Mit dem Statement SUBTRACT können Sie die Werte zweier oder mehrerer Operanden voneinander abziehen.

Operanden

Zum Format der Operanden siehe auch den Abschnitt **Formatwahl im Hinblick auf die Verarbeitungszeit** im *Natural Referenzhandbuch*.

Verwenden Sie ein Datenbankfeld als Ergebnisfeld, so ändert sich der Wert des Feldes nur programmintern. Der Wert, den das Feld in der Datenbank hat, wird davon nicht beeinflusst.

Ergebnisfeld

Wenn Sie die GIVING-Klausel verwenden, erhalten Sie das Ergebnis der Subtraktion in *operand3*; der Wert von *operand2* ändert sich nicht. Ohne GIVING-Klausel erhalten Sie das Ergebnis in *operand2*.

ROUNDED

Wünschen Sie das Ergebnis gerundet, geben Sie das Schlüsselwort **ROUNDED** an. Die für das Runden gültigen Regeln finden Sie im Abschnitt **Arithmetische Operationen** im *Natural Referenzhandbuch*.

Beispiel

```

/* EXAMPLE 'SUBEX1': SUBTRACT
/*****
DEFINE DATA LOCAL
  1 #A (P2) INIT <50>
  1 #B (P2)
  1 #C (P1.1) INIT <2.4>
END-DEFINE
/*****
SUBTRACT 6 FROM #A
WRITE NOTITLE 'SUBTRACT 6 FROM #A          ' 10X '=' #A
/*****
SUBTRACT 6 FROM 11 GIVING #A
WRITE          'SUBTRACT 6 FROM 11 GIVING #A  ' 10X '=' #A
/*****
SUBTRACT 3 4 FROM #A GIVING #B
WRITE          'SUBTRACT 3 4 FROM #A GIVING #B ' 10X '=' #A '=' #B
/*****
SUBTRACT -3 -4 FROM #A GIVING #B
WRITE          'SUBTRACT -3 -4 FROM #A GIVING #B' 10X '=' #A '=' #B
/*****
SUBTRACT ROUNDED 2.06 FROM #C
WRITE          'SUBTRACT ROUNDED 2.06 FROM #C  ' 10X '=' #C
/*****
END

```

SUBTRACT 6 FROM #A	#A: 44
SUBTRACT 6 FROM 11 GIVING #A	#A: 5
SUBTRACT 3 4 FROM #A GIVING #B	#A: 5 #B: -2
SUBTRACT -3 -4 FROM #A GIVING #B	#A: 5 #B: 12
SUBTRACT ROUNDED 2.06 FROM #C	#C: 0.3

SUSPEND IDENTICAL SUPPRESS

SUSPEND IDENTICAL [SUPPRESS] [(rep)]

Verwandte Statements

DISPLAY, WRITE.

Funktion

Mit dem Statement **SUSPEND IDENTICAL SUPPRESS** können Sie den Parameter **IS=ON** (Unterdrückung identischer Feldwerte bei der Ausgabe) für einzelne Datensätze außer Kraft setzen. Vgl. Session-Parameter **IS** im *Natural Referenzhandbuch*.

Report-Spezifikation (*rep*)

Mit der Notation “(*rep*)” kann ein bestimmter anderer Report angegeben werden, auf den sich das Statement beziehen soll. Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem **DEFINE PRINTER**-Statement zugewiesen wurde, angegeben werden.

Falls nichts anderes angegeben wird, bezieht sich das Statement **SUSPEND IDENTICAL SUPPRESS** auf den ersten Report (Report 0).

Beispiel

Programm mit SUSPEND IDENTICAL SUPPRESS:

```

/* EXAMPLE 'SISEX1': SUSPEND IDENTICAL SUPPRESS
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 FIRST-NAME
  2 NAME
  2 CITY
1 VEH-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
/*****
LIMIT 15
RD. READ EMPLOY-VIEW BY NAME STARTING FROM 'JONES'
  SUSPEND IDENTICAL SUPPRESS
FD.  FIND VEH-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (RD.)
     IF NO RECORDS FOUND
     MOVE '***NO CAR***' TO MAKE
     END-NOREC
     DISPLAY NOTITLE
           NAME (RD.) (IS=ON) FIRST-NAME (RD.) (IS=ON)
           MAKE (FD.)
     END-FIND
END-READ
END

```

NAME	FIRST-NAME	MAKE
JONES	VIRGINIA	CHRYSLER
JONES	MARSHA	CHRYSLER
JONES	ROBERT	GENERAL MOTORS
JONES	LILLY	FORD
JONES	EDWARD	MG
JONES	EDWARD	GENERAL MOTORS
JONES	MARTHA	GENERAL MOTORS
JONES	LAUREL	GENERAL MOTORS
JONES	KEVIN	DATSUN
JONES	GREGORY	FORD
JOPER	MANFRED	***NO CAR***
JOUSSELIN	DANIEL	RENAULT
JUBE	GABRIEL	***NO CAR***
JUNG	ERNST	***NO CAR***
JUNKIN	JEREMY	***NO CAR***
KAISER	REINER	***NO CAR***

Wie vorheriges Programm, aber ohne SUSPEND IDENTICAL SUPPRESS:

```

/* EXAMPLE 'SISEX2': SUSPEND IDENTICAL SUPPRESS
/* (SIMILAR TO EXAMPLE 'SISEX1' EXCEPT THAT SUSPEND IDENTICAL SUPPRESS
/* STATEMENT IS NOT USED. COMPARE OUTPUT OF EXAMPLES SISEX1 AND SISEX2
/* TO SEE EFFECT OF SUSPEND IDENTICAL).
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 FIRST-NAME
  2 NAME
  2 CITY
1 VEH-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
/*****
LIMIT 15
RD. READ EMPLOY-VIEW BY NAME STARTING FROM 'JONES'
/****SUSPEND IDENTICAL SUPPRESS STATEMENT REMOVED *****/
FD.  FIND VEH-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (RD.)
      IF NO RECORDS FOUND
        MOVE '***NO CAR***' TO MAKE
      END-NOREC
      DISPLAY NOTITLE
            NAME (RD.) (IS=ON) FIRST-NAME (RD.) (IS=ON)
            MAKE (FD.)
      END-FIND
END-READ
END

```

NAME	FIRST-NAME	MAKE
JONES	VIRGINIA	CHRYSLER
	MARSHA	CHRYSLER
		CHRYSLER
	ROBERT	GENERAL MOTORS
	LILLY	FORD
		MG
	EDWARD	GENERAL MOTORS
	MARTHA	GENERAL MOTORS
	LAUREL	GENERAL MOTORS
	KEVIN	DATSUN
	GREGORY	FORD
JOPER	MANFRED	***NO CAR***
JOUSSELIN	DANIEL	RENAULT
JUBE	GABRIEL	***NO CAR***
JUNG	ERNST	***NO CAR***
JUNKIN	JEREMY	***NO CAR***
KAISER	REINER	***NO CAR***

TERMINATE

TERMINATE [*operand1* [*operand2*]]

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	N P	ja	nein
Operand2*	C S A	A N P I F B D T L C	ja	ja

* Unter OpenVMS, UNIX und Windows muß *operand2* alphanumerisches Format haben.

Funktion

Das Statement TERMINATE bewirkt, daß die Natural-Session abgebrochen wird.

Sie können das TERMINATE-Statement an beliebiger Stelle im Programm verwenden.

Bei der Ausführung eines TERMINATE-Statements wird keine “End-of-Page”-Verarbeitung oder schleifenbeendende Verarbeitung mehr ausgeführt.

operand1

Operand1 kann dazu verwendet werden, einen Return Code an das Programm zu übergeben, das die Kontrolle erhält, nachdem die Natural-Session abgebrochen wurde. Beispielsweise könnte ein Return Code an das Betriebssystem übergeben werden (unter UNIX über “argc,argv”; unter Windows über die Betriebssystemfunktion ERRORLEVEL; auf Großrechnern über Register 15).

Für *operand1* kann ein Wert von 0 bis 255 angegeben werden.

operand2

Operand2 kann dazu verwendet werden, zusätzliche Informationen an das Programm zu übergeben, das nach dem Session-Abbruch die Kontrolle erhält.

Kontrollübergabe nach Abbruch

Nach dem Abbruch der Natural-Session erhält das Programm, dessen Name mit dem Profilparameter PROGRAM (siehe *Natural Installation and Operations Documentation*) angegeben wurde, die Kontrolle.

Unter OpenVMS und UNIX übergibt Natural die folgenden drei Parameter (falls angegeben) an dieses Programm: *operand1*, *operand2* sowie den Wert des Profilparameters PRGPAR (siehe *Natural Installation and Operations Documentation for OpenVMS and UNIX*).

Wenn unter OpenVMS der PROGRAM-Parameter nicht gesetzt ist, erhält nach dem Session-Abbruch die OpenVMS DCL die Kontrolle, und *operand1* (falls angegeben) wird an die DCL übergeben.

Wenn unter UNIX der PROGRAM-Parameter nicht gesetzt ist, erhält nach dem Session-Abbruch die UNIX Command Shell die Kontrolle, und *operand1* (falls angegeben) wird an die Command Shell übergeben.

Beispiel

```

/* EXAMPLE 'TEREX1': TERMINATE
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 SALARY (1)
1 #PNUM (A8)
1 #PASS (A8)
END-DEFINE
/*****
INPUT 'ENTER PASSWORD:' #PASS
IF #PASS NE 'USERPASS'
  TERMINATE
END-IF
/*****
INPUT 'ENTER PERSONNEL NUMBER:' #PNUM
FIND EMPLOY-VIEW WITH PERSONNEL-ID = #PNUM
  DISPLAY NAME SALARY (1)
END-FIND
/*****
END

```

UPDATE

Structured-Mode-Syntax

```
UPDATE [RECORD][IN][STATEMENT] [(r)]
```

Reporting-Mode-Syntax

```
UPDATE [RECORD][IN][STATEMENT] [(r)]
  [ SET WITH USING ] { SAME [RECORD]
    { {operand1 = operand2}... } }
```

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S A	A N P I F B D T L	nein	nein
Operand2	C S A	A N P I F B D T L	ja	nein

Verwandte Statements

STORE, DELETE, END TRANSACTION, BACKOUT TRANSACTION.

Funktion

Das Statement UPDATE dient dazu, die in der Datenbank gespeicherten Werte eines oder mehrerer Felder eines Datensatzes zu verändern.

Der betreffende Datensatz muß vorher mit einem FIND-, GET- oder READ-Statement (oder bei Adabas auch mit einem STORE-Statement) ausgewählt werden.

Hinweise für DL/I-Datenbanken

Das UPDATE-Statement kann dazu verwendet werden, ein Segment einer DL/I-Datenbank zu aktualisieren. Erforderlichenfalls wird die Länge des Segments vergrößert, um alle mit dem UPDATE-Statement angegebenen Felder aufnehmen zu können.

Ist ein multiples Feld oder eine Periodengruppe mit variabler Länge definiert, so werden nur die im UPDATE-Statement angegebenen Ausprägungen in das Segment geschrieben.

Der DL/I-AIX-Feldname darf in einem UPDATE-Statement nicht verwendet werden; AIX-Felder können aktualisiert werden, indem man sich auf das Sourcefeld bezieht, welches das betreffende AIX-Feld enthält.

DL/I-Folgefelder können aufgrund von DL/I-Beschränkungen nicht aktualisiert werden.

Aufgrund von GSAM-Beschränkungen kann das UPDATE-Statement nicht auf GSAM-Datenbanken angewandt werden.

Hinweise für SQL-Datenbanken

Mit dem UPDATE-Statement können Sie eine Reihe einer Datenbanktabelle aktualisieren. Das UPDATE-Statement entspricht dem SQL-Statement UPDATE WHERE CURRENT OF CURSOR-NAME, d.h. nur die zuletzt gelesene Reihe kann aktualisiert werden.

Auf Großrechnern werden nur Spalten (Felder) aktualisiert, die innerhalb des Programms geändert wurden, sowie Spalten, die außerhalb des Programms (z.B. als Eingabefelder in Maps) geändert worden sein könnten (aber nicht notwendigerweise auch geändert wurden). Auf allen anderen Plattformen werden alle Spalten aktualisiert.

Bei den meisten SQL-Datenbanken kann eine mit FIND SORTED BY oder READ LOGICAL gelesene Reihe nicht aktualisiert werden.

Hinweise für VSAM-Datenbanken

VSAM-Primärschlüssel können aufgrund von VSAM-Beschränkungen nicht aktualisiert werden.

Der DL/I-AIX-Feldname darf in einem UPDATE-Statement nicht verwendet werden; AIX-Felder können aktualisiert werden, indem man sich auf das Sourcefeld bezieht, welches das betreffende AIX-Feld enthält.

Einschränkungen

Das UPDATE-Statement darf nicht in derselben Sourcecode-Zeile stehen wie das Statement, mit dem der zu aktualisierende Datensatz ausgewählt wird.

Mit Entire System Server ist das UPDATE-Statement nicht verfügbar.

Statement-Referenzierung (*r*)

Mit der Notation “(*r*)” können Sie das Statement referenzieren, mit dem der Datensatz, der aktualisiert werden soll, gelesen wurde. *r* kann als Statement-Label oder Sourcecode-Zeilenummer angegeben werden.

Falls keine Referenzierung erfolgt, bezieht sich das UPDATE-Statement auf die innerste aktive READ- bzw. FIND-Verarbeitungsschleife. Ist keine READ- oder FIND-Schleife aktiv, bezieht es sich auf das letzte vorhergehende GET-Statement (bzw. STORE-Statement).

Anmerkung:

Das UPDATE-Statement muß innerhalb der READ- bzw. FIND-Schleife stehen, auf die es sich bezieht.

USING SAME

Diese Klausel ist nicht erlaubt, wenn ein DEFINE DATA-Statement verwendet wird, da sich in diesem Fall das UPDATE-Statement immer auf den gesamten im DEFINE DATA-Statement definierten View bezieht.

Mit USING SAME geben Sie im *Reporting Mode* an, daß dieselben Felder aktualisiert werden sollen, die mit dem Statement, welches vom UPDATE-Statement referenziert wird, gelesen wurden; dies bedeutet, daß zur Aktualisierung der Felder die Werte verwendet werden, die den Datenbankfeldern zuletzt zugeordnet waren. Ist kein neuer Wert zugeordnet worden, wird der alte verwendet.

Wenn das zu aktualisierende Feld ein Array-Bereich eines multiplen Feldes oder einer Periodengruppe ist und Sie einen variablen Index für diesen Array-Bereich verwenden, wird der zuletzt gültige Array-Bereich aktualisiert. Wenn die Indexvariable modifiziert wird, *nachdem* der Datensatz gelesen wurde aber *bevor* das UPDATE USING SAME- (*Reporting Mode*) bzw. UPDATE-Statement (*Structured Mode*) ausgeführt wird, bedeutet dies, daß ein anderer Array-Bereich aktualisiert wird als gelesen wurde.

SET/WITH *operand1 = operand2*

Mit dieser Klausel können Sie im *Reporting Mode* die Felder angeben, die geändert werden sollen, sowie die neuen Werte für diese Felder.

Diese Klausel ist nicht erlaubt, wenn ein DEFINE DATA-Statement verwendet wird, da sich in diesem Fall das UPDATE-Statement immer auf den gesamten im DEFINE DATA-Statement definierten View bezieht.

Anmerkung für DL/I-Datenbanken:

Wenn diese Klausel verwendet wird, dürfen nur I/O-sensitive Felder angegeben werden. Ein Segmentfolgefild darf nicht mit einem UPDATE-Statement verändert werden (verwenden Sie hierzu die Statements DELETE und STORE).

Hold-Status

Das UPDATE-Statement bewirkt, daß jeder mit dem betreffenden FIND-Statement oder READ-Statement gelesene Datensatz in den "Hold"-Status gestellt wird.

Hold-Logik ist im *Natural Leitfaden zur Programmierung* beschrieben.

Beispiel

```

/* EXAMPLE 'UPDEX1S': UPDATE (STRUCTURED MODE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
1 #NAME (A20)
END-DEFINE
/*****
INPUT 'ENTER A NAME:' #NAME (AD=M)
IF #NAME = ' '
  STOP
END-IF
/*****
FIND EMPLOY-VIEW WITH NAME = #NAME
IF NO RECORDS FOUND
  REINPUT WITH 'NO RECORDS FOUND' MARK 1
END-NOREC
INPUT 'NAME:' NAME (AD=O) /
      'FIRST NAME:' FIRST-NAME (AD=M) /
      'CITY:' CITY (AD=M)

  UPDATE
  END TRANSACTION
END-FIND
/*****
END

```

ENTER A NAME: **BROWN**

NAME: BROWN
 FIRST NAME: KENNETH
 CITY: DERBY

Äquivalentes Reporting-Mode-Beispiel: siehe Programm UPDEX1R in Library SYSEXRM.

UPLOAD

Dieses Statement ist nur mit Natural Connection verfügbar. Es ist in der *Natural Connection-Dokumentation* beschrieben.

WRITE

Syntax 1 — Dynamische Formatierung

```

WRITE [(rep)] [NOTITLE] [NOHDR]
      [(statement-parameters)]
      {
        [
          [
            nX
            nT
            x/y
            T*field-name
            P*field-name'
            /
          ] ...
        {
          'text' [(attributes)]
          'c'(n) [(attributes)]
          [=] operand1 [(parameters)]
        }
      }

```

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S A G N	A N P I F B D T L G O	ja	nein

Verwandte Statements

DISPLAY, WRITE TITLE, WRITE TRAILER, INPUT.

Funktion

Das Statement `WRITE` dient dazu, Ausgaben zu erzeugen, die nicht bereits vorformatiert sind (vgl. `DISPLAY`-Statement).

Das `WRITE`-Statement unterscheidet sich vom `DISPLAY`-Statement in folgenden Punkten:

- Paßt ein Feld bzw. Textelement nicht mehr in eine Zeile, wird es automatisch in der nächsten Zeile ausgegeben. Ein Feld bzw. Textelement wird nicht auf zwei Zeilen verteilt.
- Es werden keine Standard-Spaltenüberschriften erzeugt. Die Ausgabelänge der Felder richtet sich nach der Länge der tatsächlich ausgegebenen Feldwerte.
- Mehrere Werte/Ausprägungen eines Arrays werden nicht untereinander sondern nebeneinander ausgegeben.

Report-Spezifikation (*rep*)

Mit der Notation “(*rep*)” kann ein bestimmter Report angegeben werden, wenn ein Programm mehrere Ausgaben erzeugen soll. Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem `DEFINE PRINTER`-Statement zugewiesen wurde, angegeben werden. Wenn “(*rep*)” nicht angegeben wird, bezieht sich das `WRITE`-Statement auf den ersten Report (Report 0).

NOTITLE

Natural generiert für jede über ein `WRITE`-Statement ausgegebene Seite eine Kopfzeile. Diese Kopfzeile enthält die laufende Seitennummer, Uhrzeit und Datum. Die Uhrzeit wird zu Beginn der Programmausführung gesetzt.

Die Ausgabe dieser Standard-Kopfzeile kann durch Angabe des Schlüsselwortes `NOTITLE` unterdrückt werden. Sie können auch statt der Standard-Kopfzeile eine eigene Kopfzeile ausgeben, und zwar mit dem Statement `WRITE TITLE`.

Beispiele:

Ausgabe einer Standard-Kopfzeile:

```
WRITE NAME
```

Ausgabe einer eigenen Kopfzeile:

```
WRITE NAME  
WRITE TITLE 'USER TITLE'
```

Ausgabe ohne Kopfzeile:

```
WRITE NOTITLE NAME
```

Wenn die NOTITLE-Option verwendet wird, gilt sie für alle DISPLAY-, PRINT- und WRITE-Statements im selben Objekt, die Daten auf denselben Report schreiben.

Natural prüft, wann ein Seitenvorschub erforderlich ist, *bevor* ein WRITE-Statement ausgeführt wird. *Während* der Ausführung eines WRITE-Statements werden keine neuen Seiten mit Kopf- oder Fußzeilen generiert.

NOHDR

Das WRITE-Statement selbst erzeugt keine Spaltenüberschriften. Wenn Sie allerdings das WRITE-Statement zusammen mit einem DISPLAY-Statement verwenden, können Sie mit der Option NOHDR des WRITE-Statements die vom DISPLAY-Statement generierten Spaltenüberschriften unterdrücken: Die NOHDR-Option ist nur relevant, wenn das WRITE-Statement *nach* einem DISPLAY-Statement steht, die Ausgabe sich insgesamt über mehr als eine Seite erstreckt und die Ausführung des WRITE-Statements zur Ausgabe einer neuen Seite führt. Ohne NOHDR-Option würden auf dieser neuen Seite die DISPLAY-Spaltenüberschriften ausgegeben, mit NOHDR werden sie dort nicht ausgegeben.

statement-parameters

Unmittelbar nach dem WRITE-Statement können Sie in Klammern einzelne Session-Parameter setzen. Diese Parameterwerte haben dann für das betreffende Statement Gültigkeit vor auf übergeordneter Ebene mittels GLOBALS-Kommando, SET GLOBALS- oder FORMAT-Statement gesetzten Parameterwerten.

Wenn Sie mehrere Parameter angeben, müssen Sie sie durch ein oder mehrere Leerzeichen voneinander trennen. Eine einzelne Parameterangabe darf sich nicht über zwei Sourcecode-Zeilen erstrecken.

Die hier gültigen Parameter-Einstellungen kommen nur für Variablen-Felder in Betracht, haben aber keine Auswirkungen auf Text-Konstanten. Wenn Sie Feldattribute für eine Text-Konstante setzen möchten, dann müssen Sie explizit für dieses Element gesetzt werden.

Beispiel

```

DEFINE DATA LOCAL
1 VARI (A4)      INIT <'1234'>
END-DEFINE
*
WRITE           'Text'          VARI          /*      Output
WRITE (PM=I)   'Text'          VARI          /*      Produced
*
WRITE           'Text'          VARI          /*      -----
WRITE (PM=I)   'Text'          VARI          /*      Text 1234
WRITE           'Text' (PM=I)   VARI (PM=I) /*      Text 4321
WRITE           'Text' (PM=I)   VARI          /*      txeT 4321
END
WRITE           'Text' (PM=I)   VARI          /*      txeT 1234

```

Nähere Informationen zu den einzelnen Parametern finden Sie im Kapitel **Session-Parameter** des *Natural Referenzhandbuchs*.

Formatierung der Ausgabe

$$\left\{ \left[\begin{array}{l} nX \\ nT \\ x/y \\ T^*field-name \\ P^*field-name \\ / \\ \dots \end{array} \right] \left\{ \begin{array}{l} 'text' [(attributes)] \\ 'c'(n) [(attributes)] \\ ['='] operand1 [(parameters)] \end{array} \right\} \right\} \dots$$

Feld-Positionierung

nX

Anmerkung: (nur für Großrechner): Mit dieser Notation können Sie zwischen zwei Feldern n Leerzeichen einfügen. n darf nicht "0" sein.

Beispiel: **WRITE NAME 5X SALARY**

nT

Mit dieser Notation setzen Sie Tabulatoren, d.h. die Ausgabe eines Feldes beginnt ab Spalte n . Ein Tabulator, der bereits durch eine andere Ausgabe belegt ist, darf nicht gesetzt werden.

Beispiel: **WRITE 25T NAME 50T SALARY**

(Im obigen Beispiel wird das Feld NAME ab Spalte 25 und SALARY ab Spalte 50 ausgegeben.)

x/y

Mit dieser Notation erreichen Sie, daß ein Feld x Zeilen unter der Ausgabe des letzten Statements, und zwar ab Spalte y ausgegeben wird. y darf nicht "0" sein. Eine Spalte, die in derselben Ausgabezeile bereits belegt ist, darf nicht angegeben werden.

T**field-name*

Mit dieser Notation können Sie die WRITE-Ausgabe nach der Position eines in einem vorangegangenen DISPLAY-Statement ausgegebenen Feldes (*field-name*) ausrichten. Es ist nicht erlaubt, auf eine bereits belegte Position zu positionieren.

P**field-name*

Mit dieser Notation können Sie die WRITE-Ausgabe nach der Position *und Zeile* eines in einem vorangegangenen DISPLAY-Statement ausgegebenen Feldes (*field-name*) ausrichten. Diese Notation wird vor allem nach DISPLAY VERTICALLY-Statements verwendet. Es ist nicht erlaubt, auf eine bereits belegte Position zu positionieren.

Gleichheitszeichen '='

Ein Gleichheitszeichen in Apostrophen vor einem Feld bewirkt, daß vor dem Feldwert die (im DEFINE DATA-Statement oder im DDM) für das Feld definierte Überschrift ausgegeben wird.

Slash '/'

Ein Schrägstrich zwischen Feldern/Textelementen bewirkt einen Zeilenvorschub, d.h. die nachfolgenden Felder/ Textelemente werden in der nächsten Zeile ausgegeben.

Beispiel: **WRITE NAME / SALARY**

Für mehrfachen Zeilenvorschub geben Sie mehrere Schrägstriche an.

Text, Attribut-Zuweisung, Ausgabe-Elemente

'text'

Der in Apostrophen stehende *text* wird ausgegeben.

Beispiel: **WRITE 'EMPLOYEE' NAME 'MARITAL/STATUS' MAR-STAT**

'c'(n)

Das Zeichen *c* wird *n*-mal ausgegeben.

Beispiel: **WRITE '*' (5) '=' NAME**

attributes

Dient dazu, den auszugebenden Feldern/Texten Anzeige- und Farbattribute zuzuordnen. *Attributes* können sein:



1 Anzeigeattribute (siehe Session-Parameter AD im *Natural Referenzhandbuch*).

2 Farbattribute (siehe Session-Parameter CD im *Natural Referenzhandbuch*).

WRITE 'TEXT' (BGR)

WRITE 'TEXT' (B)

WRITE 'TEXT' (BBLC)

operand1

Der Name des auszugebenden Feldes.

Anmerkung für DL/I-Datenbanken:

DL/I-AIX-Felder können nur angezeigt werden, wenn ein PCB verwendet wird, in dem die AIX im PROCSEQ-Parameter angegeben ist. Andernfalls gibt Natural zur Laufzeit eine Fehlermeldung aus.

parameters

Unmittelbar nach *operand1* können Sie in Klammern einzelne Session-Parameter setzen. Diese Parameterwerte haben dann für das betreffende Feld Gültigkeit anstatt der mit einem GLOBALS-Kommando, SET GLOBALS- oder FORMAT-Statement oder auf Statement-Ebene gesetzten Parameterwerte. Wenn Sie mehrere Parameter angeben, müssen Sie sie durch ein oder mehrere Leerzeichen voneinander trennen. Eine Parameterangabe darf sich jeweils nicht über zwei Sourcecode-Zeilen erstrecken.

Syntax 2 — Verwendung einer vordefinierten Map

WRITE [(*rep*)] [NOTITLE] [NOHDR] [USING] { FORM
MAP } *operand1* [*operand2*...]

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S	A	nein	nein
Operand2	S A G N	A N P I F B D T L	ja	nein

FORM/MAP

Diese Option verwenden Sie, wenn Sie für die Ausgabe eine (mit dem Natural-Map-Editor erstellte) Map verwenden wollen.

WRITE USING MAP bedeutet nicht, daß jedesmal, wenn die Map ausgegeben wird, automatisch eine neue Seite ausgegeben wird.

Der Parameter LS muß um ein Byte größer gesetzt werden als die Zeilenlänge der Map.

operand1

Der Name der zu verwendenden Map.

operand2

Das auszugebende Feld.

Ist *operand1* eine Konstante und wird *operand2* nicht angegeben, so werden bei der Kompilierung die Felder aus der Map-Source übernommen.

NOTITLE/NOHDR

NOTITLE und NOHDR sind unter **Syntax 1** des WRITE-Statements beschrieben.

Beispiel 1

```

/* EXAMPLE 'WRTEX1': WRITE (USING '=', 'TEXT', '/')
/*****
LIMIT 1
READ EMPLOYEES BY NAME
/*****
WRITE NOTITLE '=' NAME '=' FIRST-NAME '=' MIDDLE-I //
          'L O C A T I O N' /
          'CITY:' CITY      /
          'COUNTRY:' COUNTRY //
/*****
END

```

NAME: ABELLAN	FIRST-NAME: KEPA	MIDDLE-I:
L O C A T I O N		
CITY: MADRID		
COUNTRY: E		

Beispiel 2

```

/* EXAMPLE 'WRTEX2:' WRITE (USING NX, NT NOTATION)
/*****
LIMIT 4
READ EMPLOYEES BY NAME
  WRITE NOTITLE 5X NAME  50T JOB-TITLE
/*****
END

```

ABELLAN	MAQUINISTA
ACHIESON	DATA BASE ADMINISTRATOR
ADAM	CHEF DE SERVICE
ADKINSON	SALES PERSON

Beispiel 3

```

/* EXAMPLE 'WRTEX3': WRITE (USING T* NOTATION)
/*****
LIMIT 5
READ EMPLOYEES BY CITY STARTING FROM 'ALBU'
  DISPLAY NOTITLE CITY NAME SALARY (1)
  AT BREAK CITY
/*****
  WRITE / 'CITY AVERAGE:' T*SALARY (1) AVER(SALARY(1)) //
/*****
END

```

CITY	NAME	ANNUAL SALARY
ALBUQUERQUE	HAMMOND	22000
ALBUQUERQUE	ROLLING	34000
ALBUQUERQUE	FREEMAN	34000
ALBUQUERQUE	LINCOLN	41000
CITY AVERAGE:		32750
ALFRETON	GOLDBERG	4700
CITY AVERAGE:		4700

Beispiel 4

```

* EXAMPLE 'WRTEX4': WRITE (USING P* NOTATION)
*****
LIMIT 3
READ EMPLOYEES BY CITY FROM 'N'
  DISPLAY NOTITLE NAME CITY
    VERT AS 'BIRTH/SALARY' BIRTH (EM=YYYY-MM-DD) SALARY (1)
  SKIP 1
  AT BREAK CITY
*
  WRITE / 'CITY AVERAGE' P*SALARY (1) AVER(SALARY (1)) //
*
LOOP
END

```

NAME	CITY	BIRTH SALARY
WILCOX	NASHVILLE	1970-01-01 38000
MORRISON	NASHVILLE	1949-07-10 36000
CITY AVERAGE		37000
BOYER	NEMOURS	1955-11-23 195900
CITY AVERAGE		195900

Beispiel 5

```
/* EXAMPLE 'WRTEX5': WRITE (USING '=', STATEMENT/ELEMENT PARAMETERS)
/*****
LIMIT 2
READ EMPLOYEES BY NAME
  WRITE NOTITLE (AL=16 NL=8)
    '=' PERSONNEL-ID '=' NAME '=' PHONE (AL=10 EM=XXX-XXXXXXX)
/*****
END
```

PERSONNEL ID: 60008339	NAME: ABELLAN	TELEPHONE: 435-672
PERSONNEL ID: 30000231	NAME: ACHIESON	TELEPHONE: 523-341

WRITE TITLE

WRITE [(rep)] **TITLE** [LEFT [JUSTIFIED]] [UNDERLINED]

[(statement-parameters)]

$$\left\{ \left[\begin{array}{l} nX \\ nT \\ x/y \end{array} \right] \dots \left\{ \begin{array}{l} \text{'text' [(attributes)]} \\ \text{'c'(n) [(attributes)]} \\ \text{['='] operand1 [(parameters)]} \end{array} \right\} \right\} \dots$$

[SKIP operand2 [LINES]]

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S A G N	A N P I F B D T L G O	ja	nein
Operand2	C S	N P I B	ja	nein

Verwandte Statements

WRITE, DISPLAY, WRITE TRAILER.

Funktion

Das Statement WRITE TITLE dient dazu, statt einer Standard-Kopfzeile eine eigene Seitenüberschrift auszugeben. Das Statement wird immer dann ausgeführt, wenn eine neue Ausgabe-seite initiiert wird.

Dieses Statement ist non-prozedural (das heißt, seine Ausführung hängt von einem Ereignis ab, nicht davon, wo im Programm es steht).

Anmerkung:

Wenn ein Report durch Statements in verschiedenen Objekten erzeugt wird, wird das WRITE TITLE-Statement nur ausgeführt, wenn es in demselben Objekt steht wie das Statement, das die Ausgabe einer neuen Seite auslöst.

Einschränkungen

WRITE TITLE darf höchstens einmal pro Ausgabe-Report verwendet werden.

WRITE TITLE darf nicht an eine logische Bedingung geknüpft sein.

WRITE TITLE darf nicht in einer Subroutine stehen.

Report-Spezifikation (*rep*)

Erzeugt ein Programm mehrere Reports, kann mit der Notation “(*rep*)” ein bestimmter anderer Report angegeben werden, auf den sich das Statement beziehen soll. Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden.

Wenn “(*rep*)” nicht angegeben wird, bezieht sich das Statement WRITE TITLE auf den ersten Report (Report 0).

Linksbündige Ausrichtung und Unterstreichung

Normalerweise werden Seitenkopfzeilen zentriert und ohne Unterstreichung ausgegeben. Eine linksbündige Ausrichtung der Kopfzeile erreichen Sie durch die Angabe des Schlüsselwortes LEFT JUSTIFIED. Eine unterstrichene Kopfzeile erhalten Sie durch Angabe des Schlüsselwortes UNDERLINED; als Unterstreichungszeichen wird das mit dem Parameter UC (auf Session-Ebene oder in einem FORMAT-Statement) definierte Zeichen verwendet. Die Unterstreichung erstreckt sich über die ganze Zeile unter der Kopfzeile (entsprechend der mit dem Parameter LS definierten Zeilenlänge).

Bevor die Zeile zentriert wird, führt Natural erst alle Leerstellen- und Tabulatoranweisungen aus. Bei einer Tabulator-Notation von “10T”, zum Beispiel, rückt der Text bei anschließender Zentrierung in eine Position 5 Stellen rechts von der Mitte.

statement-parameters

Unmittelbar nach dem WRITE TITLE-Statement können Sie in Klammern einzelne Session-Parameter setzen. Diese Parameterwerte haben dann für das betreffende Statement Gültigkeit vor auf übergeordneter Ebene mittels GLOBALS-Kommando, SET GLOBALS- oder FORMAT-Statement gesetzten Parameterwerten.

Wenn Sie mehrere Parameter angeben, müssen Sie sie durch ein oder mehrere Leerzeichen voneinander trennen. Eine einzelne Parameterangabe darf sich nicht über zwei Sourcecode-Zeilen erstrecken.

Eine Beschreibung der einzelnen Parameter finden Sie im Kapitel **Session-Parameter** des *Natural Referenzhandbuchs*.

operand1

Als *operand1* können Sie ein oder mehrere Feld/er angeben, die in der Kopfzeile ausgegeben werden sollen. Die Format- und Leerschritt-Notationen entsprechen den im WRITE-Statement verwendeten (weitere Informationen siehe WRITE-Statement).

SKIP (*operand2*)

Mit der SKIP-Klausel können Sie nach der Kopfzeile Leerzeilen einfügen. Mit *operand2* geben Sie an, wieviele Leerzeilen auf die Kopfzeile folgen sollen; dies kann entweder eine numerische Konstante oder der Inhalt einer numerischen Variablen sein.

Anmerkung:

SKIP nach WRITE TITLE wird immer als SKIP-Klausel des WRITE TITLE-Statements interpretiert, und nicht als eigenständiges Statement. Falls Sie ein eigenständiges SKIP-Statement nach einem WRITE TITLE-Statement wünschen, trennen Sie die beiden Statements durch ein Semikolon (;) voneinander.

Beispiel

```

/* EXAMPLE 'WTIEX1': WRITE TITLE
/*****
FORMAT LS=70
/*****
WRITE TITLE LEFT JUSTIFIED UNDERLINED
      *TIME 3X 'PEOPLE LIVING IN NEW YORK CITY'
      11X 'PAGE:' *PAGE-NUMBER

SKIP1
/*****
FIND EMPLOYEES WITH CITY = 'NEW YORK'
  DISPLAY NAME FIRST-NAME 3X JOB-TITLE
/*****
END

```

15:57:19.1	PEOPLE LIVING IN NEW YORK CITY	PAGE:	1
<hr/>			
NAME	FIRST-NAME	CURRENT POSITION	
<hr/>			
RUBIN	SYLVIA	SECRETARY	
WALLACE	MARY	ANALYST	

WRITE TRAILER

WRITE [(*rep*)] **TRAILER** [LEFT [JUSTIFIED]] [UNDERLINED]

[(*statement-parameters*)]

$$\left\{ \left[\begin{array}{l} nX \\ nT \\ x/y \end{array} \right] \dots \left\{ \begin{array}{l} 'text' [(attributes)] \\ 'c'(n) [(attributes)] \\ ['='] operand1 [(parameters)] \end{array} \right\} \right\} \dots$$

[SKIP *operand2* [LINES]]

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	S A G N	A N P I F B D T L G O	ja	nein
Operand2	C S	N P I B	ja	nein

Verwandte Statements

WRITE, DISPLAY, WRITE TITLE.

Funktion

Das Statement WRITE TRAILER dient dazu, am Ende einer Ausgabeseite eine Fußzeile auszugeben.

Dieses Statement ist non-prozedural (das heißt, seine Ausführung hängt von einem Ereignis ab, nicht davon, wo im Programm es steht).

Einschränkungen

WRITE TRAILER darf höchstens einmal pro Ausgabe-Report verwendet werden.

WRITE TRAILER darf nicht an eine logische Bedingung geknüpft sein.

WRITE TRAILER darf nicht in einer Subroutine stehen.

Verarbeitung

Das Statement wird immer dann ausgeführt, wenn eine “End-of-Page”- oder “End-of-Data”-Bedingung auftritt, oder wenn aufgrund eines SKIP- oder NEWPAGE-Statements ein Seitenvorschub erfolgt. Es wird *nicht* ausgeführt, wenn ein Seitenvorschub aufgrund eines EJECT-Statements erfolgt.

Ob eine “End-of-Page”-Bedingung gegeben ist, wird erst überprüft, *nachdem* ein DISPLAY- bzw. WRITE-Statement vollständig ausgeführt ist; ist die logische Seitenlänge nicht richtig gesetzt, kann es vorkommen, daß die DISPLAY-/WRITE-Ausgabe bereits das Ende einer physischen Ausgabeseite überschritten hat, bevor auf der logischen Seite eine “End-of-Page”-Bedingung auftritt.

Anmerkung:

Wenn ein Report durch Statements in verschiedenen Objekten erzeugt wird, wird das WRITE TRAILER-Statement nur ausgeführt, wenn es in demselben Objekt steht wie das Statement, das die “End-of-Page”-Bedingung auslöst.

Logische Seitenlänge

Um sicherzustellen, daß eine mit WRITE TRAILER definierte Fußzeile noch auf eine ausgegebene physische Seite paßt, sollte die Länge der vom Programm erzeugten logischen Seite (mittels des Parameters PS) entsprechend kleiner als die physische Seitenlänge gesetzt werden.

Report-Spezifikation (*rep*)

Mit der Notation “(*rep*)” kann ein bestimmter anderer Report angegeben werden, auf den sich das Statement beziehen soll. Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden.

Wenn “(*rep*)” nicht angegeben wird, so bezieht sich das Statement WRITE TRAILER auf den ersten Report (Report 0).

Linksbündige Ausrichtung und Unterstreichung

Normalerweise werden Fußzeilen zentriert und ohne Unterstreichung ausgegeben. Eine linksbündige Ausrichtung der Fußzeile erreichen Sie durch die Angabe des Schlüsselwortes LEFT JUSTIFIED. Eine unterstrichene Fußzeile erhalten Sie durch Angabe des Schlüsselwortes UNDERLINED; als Unterstreichungszeichen wird das mit dem Parameter UC (auf Session-Ebene oder in einem FORMAT-Statement) definierte Zeichen verwendet. Die Unterstreichung erstreckt sich über die ganze Zeile unter der Fußzeile (entsprechend der mit dem Parameter LS definierten Zeilenlänge).

Bevor die Zeile zentriert wird, führt Natural erst alle Leerstellen- und Tabulatoranweisungen aus. Bei einer Tabulator-Notation von “10T”, zum Beispiel, rückt der Text bei anschließender Zentrierung in eine Position 5 Stellen rechts von der Mitte.

statement-parameters

Unmittelbar nach dem WRITE TRAILER-Statement können Sie in Klammern einzelne Session-Parameter setzen. Diese Parameterwerte haben dann für das betreffende Statement Gültigkeit vor auf übergeordneter Ebene mittels GLOBALS-Kommando, SET GLOBALS- oder FORMAT-Statement gesetzten Parameterwerten.

Wenn Sie mehrere Parameter angeben, müssen Sie sie durch ein oder mehrere Leerzeichen voneinander trennen. Eine einzelne Parameterangabe darf sich nicht über zwei Sourcecode-Zeilen erstrecken.

Eine Beschreibung der einzelnen Parameter finden Sie im Kapitel **Session-Parameter** des *Natural Referenzhandbuchs*.

operand1

Als *operand1* können Sie ein oder mehrere Feld/er angeben, die in der Fußzeile ausgegeben werden sollen.

Die Format- und Leerschritt-Notationen entsprechen den im WRITE-Statement verwendeten (weitere Informationen siehe WRITE-Statement).

SKIP (*operand2*)

Mit der SKIP-Klausel können Sie nach der Fußzeile Leerzeilen einfügen. Mit *operand2* geben Sie an, wieviele Leerzeilen auf die Fußzeile folgen sollen; dies kann entweder eine numerische Konstante oder der Inhalt einer numerischen Variablen sein.

Anmerkung:

SKIP nach WRITE TRAILER wird immer als SKIP-Klausel des WRITE TRAILER-Statements interpretiert, und nicht als eigenständiges Statement. Falls Sie ein eigenständiges SKIP-Statement nach einem WRITE TRAILER-Statement wünschen, trennen Sie die beiden Statements durch ein Semikolon (;) voneinander.

Beispiel

```

/* EXAMPLE 'WTLEX1': WRITE TRAILER
/*****
FORMAT PS=15
WRITE TITLE LEFT JUSTIFIED UNDERLINED
      *TIME 3X 'PEOPLE LIVING IN BARCELONA'
      14X 'PAGE:' *PAGE-NUMBER
      SKIP 1
/*****
WRITE TRAILER LEFT JUSTIFIED UNDERLINED
  / 'CITY OF BARCELONA REGISTER'
/*****
LIMIT 10
FIND EMPLOYEES WITH CITY = 'BARCELONA'
  DISPLAY NAME FIRST-NAME 3X JOB-TITLE
/*****
END

```

NAME	FIRST-NAME	CURRENT POSITION
DEL CASTILLO	ANGEL	EJECUTIVO DE VENTAS
GARCIA	M. DE LAS MERCEDES	SECRETARIA
GARCIA	ENDIKA	DIRECTOR TECNICO
MARTIN	ASUNCION	SECRETARIA
MARTINEZ	TERESA	SECRETARIA
YNCLAN	FELIPE	ADMINISTRADOR
FERNANDEZ	ELOY	OFICINISTA
TORRES	ANTONI	OBRERA
CITY OF BARCELONA REGISTER		

NAME	FIRST-NAME	CURRENT POSITION
RODRIGUEZ	VICTORIA	SECRETARIA
GARCIA	GERARDO	INGENIERO DE PRODUCCION
CITY OF BARCELONA REGISTER		

WRITE WORK FILE

WRITE WORK [FILE] *work-file-number* [VARIABLE] *operand1...*

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
Operand1	C S A G	A N P I F B D T L C G	ja	nein

Verwandte Statements

DEFINE WORK FILE, READ WORK FILE, CLOSE WORK FILE.

Funktion

Das Statement WRITE WORK FILE dient dazu, Datensätze auf eine physisch-sequentielle Arbeitsdatei (Work File) zu schreiben.

Auf Großrechnern kann dieses Statement nur im Batch-Betrieb oder unter Com-plete, CMS, TSO und TIAM verwendet werden.

Es ist möglich, in einem Programm oder einer Verarbeitungsschleife eine Arbeitsdatei zu erstellen und diese dann in einem anderen Programm oder einer anderen eigenständigen Verarbeitungsschleife mit einem READ WORK FILE-Statement zu lesen.

work-file-number

Die für Natural definierte Nummer der Arbeitsdatei, die verwendet werden soll.

VARIABLE

Anmerkung:

Diese Option steht nur auf Großrechnern zur Verfügung.

Es ist möglich, mittels verschiedener WRITE WORK FILE-Statements Datensätze mit verschiedenen Feldern auf dieselbe Arbeitsdatei zu schreiben. In diesem Fall müssen alle betreffenden WRITE WORK FILE-Statements das Schlüsselwort VARIABLE enthalten; die Datensätze werden dann mit variablem Format auf die externe Datei geschrieben. Natural schreibt alle Ausgabedateien in variablen Blöcken (es sei denn, Sie geben in der Ausführungs-JCL Datensatzformat und Blockgröße an; nur möglich auf Großrechnern).

Felder (*operand1*)

Als *operand1* geben Sie die Felder an, die auf die Arbeitsdatei geschrieben werden sollen. Dies können entweder Datenbankfelder, Benutzervariablen oder Felder sein, die mit einem READ WORK FILE-Statement von einer anderen Arbeitsdatei gelesen wurden.

Bei Datenbank-Arrays kann durch eine einen Bereich umfassende Indexierung angegeben werden, welche Ausprägungen auf die Arbeitsdatei geschrieben werden sollen. Feldgruppen können durch Angabe des Gruppennamens referenziert werden; alle Felder einer Gruppe werden einzeln auf die Arbeitsdatei geschrieben.

Variabler Indexbereich

Wenn Sie ein Array auf eine Arbeitsdatei schreiben, können Sie für das Array einen variablen Indexbereich angeben. Zum Beispiel:

WRITE WORK FILE *work-file-number* **VARIABLE #ARRAY (I:J)**

Externe Darstellung der Felder

Auf eine Arbeitsdatei geschriebene Felder werden auf der externen Datei entsprechend ihrer internen Definition dargestellt. Die Feldwerte werden nicht verändert.

Bei Feldern der Formate A oder B entspricht die Anzahl der Bytes auf der externen Datei der programminternen Längendefinition. Die Feldwerte werden nicht verändert; ein Komma (Dezimalpunkt) wird nicht wiedergegeben.

Bei Feldern des Formats N ergibt sich die Anzahl der Bytes auf der externen Datei aus der Summe der Stellen vor und nach dem Komma. Das Komma (Dezimalpunkt) wird auf der externen Datei nicht wiedergegeben.

Bei Feldern des Formats P ergibt sich die Anzahl der Bytes auf der externen Datei aus der Summe der Stellen vor und nach dem Komma plus einer Stelle für das Vorzeichen, geteilt durch 2, wobei auf ganze Bytes aufgerundet wird.

Anmerkung:

Beim Schreiben von Feldern auf eine Arbeitsdatei erfolgt keine Umsetzung von Feldformaten.

Beispiele für Felddarstellung:

Felddefinition	Ausgabelänge
#FIELD1 (A10)	10 Bytes
#FIELD2 (B15)	15 Bytes
#FIELD3 (N1.3)	4 Bytes
#FIELD4 (N0.7)	7 Bytes
#FIELD5 (P1.2)	2 Bytes
#FIELD6 (P6.0)	4 Bytes

Anmerkung:

Wenn die Systemfunktionen *AVER*, *NAVER*, *SUM* oder *TOTAL* für numerische Felder (Format *N* oder *P*) auf eine Arbeitsdatei geschrieben werden, vergrößert sich intern die Länge dieser Felder um eine Stelle (z.B.: *SUM* eines Feldes vom Format *P3* wird auf *P4* verlängert). Dies ist beim Lesen der Arbeitsdatei zu berücksichtigen.

Verarbeitung großer und dynamischer Variablen

Die RECORD-Option ist nicht zulässig, wenn dynamische Variablen verwendet werden.

Bei den Arbeitsdateitypen ASCII, ASCII-COMPRESSED, ENTIRECONNECTION, SAG (binary) und TRANSFER können dynamische Variablen nicht verarbeitet werden, was zu einem Fehler führt. Große Variablen stellen überhaupt kein Problem dar, außer wenn die maximale Feld/Datensatzlänge überschritten wird (Feldlänge 255 für ENTIRECONNECTION und TRANSFER, Datensatzlänge 32767 für die anderen).

Der Arbeitsdateityp PORTABLE speichert die Feldinformationen in der Arbeitsdatei, so daß die Länge dynamischer Variablen bei einer READ-Operation verändert wird, wenn die Feldlänge im Datensatz nicht mit der aktuellen Länge identisch ist.

Beispiel

```
/* EXAMPLE 'WWFEX1': WRITE WORK FILE
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
END-DEFINE
/*****
FIND EMPLOY-VIEW WITH CITY = 'LONDON'
  WRITE WORK FILE 1
    PERSONNEL-ID
    NAME
END-FIND
/*****
END
```


SQL-STATEMENTS

Alte Statements

Im folgenden finden Sie eine Liste alter Natural-Statements.

Sie können in Verbindung mit Natural Expert und Predict Case verwendet werden.

- DLOGOFF/DLOGON
- SHOW
- IMPORT
- EXPORT
- INVESTIGATE

Weitere Informationen über diese Statements entnehmen Sie der *Natural Expert* und *Predict Case Documentation*.

SQL-Statements-Übersicht

Neben den im Kapitel **Statements** beschriebenen “eigentlichen” Natural-Statements bietet Natural SQL-Statements, so daß Sie in Natural-Programmen SQL direkt benutzen können.

Folgende SQL-Statements sind verfügbar und in diesem Kapitel beschrieben:

- **CALLDBPROC** (Seite 716)
- **COMMIT** (Seite 721)
- **DELETE** (Seite 722)
- **INSERT** (Seite 724)
- **PROCESS SQL** (Seite 727)
- **READ RESULT SET** (Seite 731)
- **ROLLBACK** (Seite 733)
- **SELECT** (Seite 735)
- **UPDATE** (Seite 749).

Anmerkung:

Bezüglich der Portabilität von Natural-Anwendungen beachten Sie bitte, daß die Natural-SQL-Statements nur für Datenbanksysteme verwendet werden können, die SQL unterstützen, wohingegen Natural DML-Statements wie FIND und READ für alle von Natural unterstützten Datenbanksysteme verwendet werden können.

Dieses Kapitel behandelt die folgenden Themen: Common Set und Extended Set.

Vor der Beschreibung der eigentlichen Statements sind folgende Punkte — die mehrere SQL-Statements betreffen — in diesem Kapitel beschrieben:

- grundlegende Syntaxbestandteile (Seite 679)
- das Natural-View-Konzept (Seite 690)
- “Scalar Expressions” (Seite 692)
- “Search Conditions” (Seite 698)
- “Select Expressions” (Seite 705).

Weiter vorne in diesem Kapitel (Seite 712) ist eine weitere Möglichkeit beschrieben, SQL-Statements einzusetzen: die sogenannte “flexible SQL”, mit der Sie die Möglichkeit haben, beliebige SQL-Syntax zu verwenden.

Common Set und Extended Set

Die in Natural verfügbaren SQL-Statements umfassen zwei Syntax-Gruppen:

- eine allgemeine (“Common Set”) und
Der “Common Set” entspricht im Prinzip den Syntaxdefinitionen der Standard-SQL und kann für alle von Natural unterstützten SQL-fähigen Datenbanksysteme verwendet werden und
- eine erweiterte (“Extended Set”)
Der “Extended Set” bietet darüber hinaus einige spezielle Erweiterungen zur Unterstützung von bestimmten Funktionen verschiedener von Natural unterstützter Datenbanksysteme. Die zur Verfügung stehenden Teile des Extended Set sind von Datenbanksystem zu Datenbanksystem unterschiedlich..

Dieses Kapitel beschreibt in erster Linie den Common Set. Die Statement-Syntax entspricht weitestmöglich der in der betreffenden SQL-Literatur beschriebenen; Einzelheiten finden Sie in dieser Literatur. Einzelheiten zum Extended Set finden Sie in der Dokumentation der Natural-Schnittstelle zu dem jeweiligen von Ihnen verwendeten Datenbanksystem.

Die nur im Rahmen des *Extended Set* verfügbaren Syntaxteile sind in den Syntaxdiagrammen in diesem Kapitel *grau unterlegt*.

Grundlegende Syntaxbestandteile

In diesem Abschnitt sind grundlegende Syntaxbestandteile beschrieben, die dann in den Beschreibungen der einzelnen Statements nicht mehr näher erläutert werden. Diese Teile sind:

- Konstanten
- Namen
- Parameter.

Konstanten

Die in den Syntaxbeschreibungen von Natural-SQL-Statements verwendeten Konstanten sind *constant* und *integer*.

Anmerkung:

Wenn das Dezimalzeichen (Session Parameter DC) auf Komma (,) gesetzt ist, darf unmittelbar nach einer numerischen Konstanten kein Komma angegeben werden, sondern es muß ein Leerzeichen dazwischen stehen, da es sonst zu Fehlern oder falschen Ergebnissen kommen kann.

Falsche Beispiele:

VALUES (1,'A')
VALUES (1,2,3)

führt zu Syntaxfehler
führt zu falschen Ergebnissen

Richtige Beispiele:

VALUES (1 , 'A')
VALUES (1 ,2 ,3)

constant

Bei *constant* handelt es sich immer um eine Natural-Konstante.

integer

Bei *integer* handelt es sich immer um eine Ganzzahl-Konstante.

Namen

Die in den Syntaxbeschreibungen von Natural-SQL-Statements verwendeten Namen sind *authorization-identifier*, *ddm-name*, *view-name*, *column-name*, *table-name*, *correlation-name*.

authorization-identifier

Ein *authorization-identifier*, der auch “creator name” genannt wird, dient zur Qualifizierung von Datenbanktabellen und Views.

ddm-name

ddm-name ist jeweils der Name eines mit der Natural-Utility SYSDDM erzeugten Natural-DDMs.

view-name

view-name ist jeweils der Name eines im DEFINE DATA-Statement definierten Views.

column-name

column-name ist jeweils der Name einer physischen Datenbankspalte.

table-name

<i>[authorization-identifier.] ddm-name</i>

Table-name in diesem Kapitel dient zur Referenzierung von SQL-Basistabellen und SQL-“Viewed“-Tabellen. Für jede Tabelle muß ein entsprechendes Natural-DDM existieren. Der Name des DDMS muß mit dem Namen der entsprechenden physischen Datenbanktabelle bzw. des Views identisch sein.

authorization-identifier

Es gibt zwei Arten, den *authorization-identifier* einer Datenbanktabelle bzw. eines Views anzugeben.

Die eine Art entspricht der Standard-SQL-Syntax: *authorization-identifier* und DDM-Name werden durch einen Punkt miteinander verbunden. Hierbei muß der DDM-Name dem Namen der physischen Datenbanktabelle (ohne *authorization-identifier*) entsprechen.

Beispiel:

```
DEFINE DATA LOCAL
01 PERS VIEW OF PERSONNEL
    02 NAME
    02 AGE
END-DEFINE
SELECT *
    INTO VIEW PERS
    FROM SQL.PERSONNEL
...
```

Die andere Möglichkeit besteht darin, den *authorization-identifier* als Teil des DDM-Namens selbst zu definieren. Der DDM-Name besteht dann aus dem *authorization-identifier* gefolgt von einem Bindestrich (-) und gefolgt vom Namen der Datenbanktabelle. Intern wird der Bindestrich zwischen *authorization-identifier* und Tabellennamen in einen Punkt umgesetzt.

Anmerkung:

Diese Form des DDM-Namens kann auch in einem FIND- oder READ-Statement verwendet werden, da sie den für diese Statements geltenden DDM-Namenskonventionen entspricht.

Beispiel:

```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
    02 NAME
    02 AGE
END-DEFINE
SELECT *
    INTO VIEW PERS
    FROM SQL-PERSONNEL
...
```

Wenn der *authorization-identifier* weder explizit noch als Teil des DDM-Namens angegeben wird, wird er vom betreffenden SQL-Datenbanksystem bestimmt.

Table-names können nicht nur in SELECT-Statements, sondern auch in DELETE-, INSERT- und UPDATE-Statements verwendet werden.

Beispiele:

```
...
DELETE FROM SQL.PERSONNEL
  WHERE AGE IS NULL
...

...
INSERT INTO SQL.PERSONNEL (NAME,AGE)
  VALUES ('ADKINSON',35)
...

...
UPDATE SQL.PERSONNEL
SET SALARY = SALARY * 1.1
WHERE AGE > 30
...
```

correlation-name

Correlation-name ist ein Alias-Name für einen *table-name*. Er kann zur Qualifizierung von Spaltennamen verwendet werden. Außerdem dient er dazu, implizit Felder in einem Natural-View zu qualifizieren, der in der INTO-Klausel eines SELECT-Statements verwendet wird (vgl. SELECT-Statement, Seite 735).

Beispiel:

```
DEFINE DATA LOCAL
01 PERS-NAME      (A20)
01 EMPL-NAME      (A20)
01 AGE            (I2)
END-DEFINE
...
SELECT X.NAME , Y.NAME , X.AGE
      INTO PERS-NAME , EMPL-NAME , AGE
      FROM SQL-PERSONNEL X , SQL-EMPLOYEES Y
      WHERE X.AGE = Y.AGE
END-SELECT
...
```

Die Verwendung von *correlation-names* ist zwar in der Regel nicht nötig, kann aber helfen, die Lesbarkeit eines Statements zu erleichtern.

Parameter

parameter

`[:]host-variable [INDICATOR [:]host-variable] [LINDICATOR [:]host-variable]`

host-variable

Eine *host-variable* ist eine in einem SQL-Statement referenzierte Natural-Programmvariable, die entweder ein eigenständiges Feld oder Teil eines Views sein kann.

Wenn sie als empfangendes Feld (z.B. in einer INTO-Klausel) definiert wird, ist die *host-variable* ein Feld, das vom Datenbanksystem einen Wert erhält.

Wenn sie als sendendes Feld (z.B. in einer WHERE-Klausel) definiert wird, ist die *host-variable* ein Feld, dessen Wert vom Programm an das Datenbanksystem übergeben wird.

colon[:]

Gemäß den SQL-Standards kann einer *host-variable* ein Doppelpunkt (:) vorangestellt werden. Bei der Verwendung mit flexibler SQL *muß* ihr ein Doppelpunkt vorangestellt werden (vgl. Seite 712).

Beispiel:

```
SELECT NAME INTO :#NAME FROM PERSONNEL
WHERE AGE = :VALUE
```

Wenn ein Variablenname mit einem für SQL reservierten Wort identisch ist, ist der Doppelpunkt ebenfalls erforderlich. In Situationen, in denen entweder eine *host-variable* oder eine Spalte referenziert werden kann, wird ein Name ohne Doppelpunkt als Spaltenname interpretiert.

Natural-Formate und SQL-Datentypen

Das Natural-Format einer *host-variable* wird entsprechend der folgenden Tabelle in einen SQL-Datentyp umgesetzt:

Natural-Format/Länge	SQL-Datentyp
<i>An</i>	CHAR (<i>n</i>)
B2	SMALLINT
B4	INT
<i>Bn</i> ; <i>n</i> ungleich 2 oder 4	CHAR (<i>n</i>)
F4	REAL
F8	DOUBLE PRECISION
I2	SMALLINT
I4	INT
<i>Nnn.m</i>	NUMERIC (<i>nn+m,m</i>)
<i>Pnn.m</i>	NUMERIC (<i>nn+m,m</i>)
<i>Gn</i> ; nur für Views	GRAPHIC (<i>n</i>)

Natural überprüft nicht, ob der SQL-Datentyp mit der Datenbankspalte kompatibel ist. Außer bei Feldern vom Format N wird keine Datenkonvertierung vorgenommen.

Mit Natural SQL gibt es zu den Standard-Natural-Formaten noch folgende Erweiterungen:

- Um alphanumerische Spalten zu unterstützen, die länger als 253 Bytes sind, kann ein eindimensionales Array vom Format A verwendet werden. Der Index dieses Arrays muß mit 1 anfangen und kann nur mit (*) referenziert werden. Der entsprechende SQL-Datentyp ist CHAR (n), wobei n die Gesamtanzahl der Bytes des Arrays ist.
- Um Spalten mit variabler Länge zu unterstützen, kann eine *host-variable* mit Schlüsselwort LINDICATOR verwendet werden. Der entsprechende SQL-Datentyp ist VARCHAR (n); vgl. **LINDICATOR-Klausel** (Seite 688).
- Die Natural-Formate Datum (D) und Zeit (T) können mit Entire Access verwendet werden und werden in die entsprechenden datenbank-spezifischen Formate umgesetzt (Näheres siehe *Entire Access-Dokumentation*).

Ein sendendes Feld, das als eindimensionales Array ohne LINDICATOR-Feld angegeben wird, wird in den SQL-Datentyp VARCHAR umgesetzt. Seine Länge ist die Gesamtanzahl der Bytes des Arrays ohne Berücksichtigung nachgestellter Leerzeichen.

INDICATOR-Klausel

Die INDICATOR-Klausel ist optional und dient dazu, herauszufinden, ob eine zu lesende Spalte “null” ist, d.h. *keinen* Wert enthält, oder tatsächlich den Wert “0” bzw. Leerzeichen enthält.

Wenn sie mit einer empfangenden *host-variable* (Zielfeld) verwendet wird, dient die INDICATOR *host-variable* (Null-Indikatorfeld) dazu, herauszufinden, ob eine zu lesende Spalte “null” ist.

Beispiel:

```
DEFINE DATA LOCAL
1 NAME      (A20)
1 NAMEIND   (I2)
END-DEFINE
SELECT *
  INTO NAME INDICATOR NAMEIND
  ...
```

In diesem Beispiel ist NAME die empfangende *host-variable* und NAMEIND das Null-Indikatorfeld.

Ist ein Null-Indikatorfeld angegeben und die gelesene Spalte ist null, wird das Indikatorfeld auf einen negativen Wert und das Zielfeld je nach Datentyp auf “0” bzw. Leerzeichen gesetzt. Andernfalls ist der Wert des Null-Indikatorfeldes größer gleich “0”.

Wenn sie mit einer sendenden *host-variable* (Ausgangsfeld) verwendet wird, dient die INDICATOR *host-variable* dazu, dem Ausgangsfeld einen Nullwert zuzuweisen.

Beispiel:

```
DEFINE DATA LOCAL
1 NAME      (A20)
1 NAMEIND   (I2)
UPDATE ...
SET NAME = :NAME INDICATOR :NAMEIND
WHERE ...
```

In diesem Beispiel ist :NAME die sendende *host-variable* und :NAMEIND das Null-Indikatorfeld. Durch Eingabe eines negativen Wertes in das Null-Indikatorfeld wird der Datenbankspalte ein Nullwert zugewiesen.

Eine INDICATOR *host-variable* hat Format/Länge I2.

LINDICATOR-Klausel

Die LINDICATOR-Klausel ist optional und dient zur Unterstützung von Spalten des Typs VARCHAR oder LONG VARCHAR, die bis zu 32 KB lang sein können.

Wenn sie mit einer empfangenden *host-variable* (Zielfeld) verwendet wird, enthält die LINDICATOR *host-variable* (Längen-Indikatorfeld) die Anzahl der tatsächlich von der Datenbank in das Zielfeld geschriebenen Zeichen. Das Zielfeld wird immer mit Leerzeichen aufgefüllt.

Enthält die VARCHAR- bzw. LONG VARCHAR-Spalte mehr Zeichen als in das Zielfeld passen, wird im Längen-Indikatorfeld die Anzahl der tatsächlich gelesenen Zeichen ausgegeben und im Null-Indikatorfeld (falls angegeben) die tatsächliche Gesamtlänge der Spalte.

Beispiel:

```
DEFINE DATA LOCAL
1 ADDRESSIND (I2)
1 ADDRESS (A50/1:6)
END-DEFINE
SELECT *
  INTO :ADDRESS(*) LINDICATOR :ADDRESSLIND
  ...
```

:ADDRESS(*) erhält die ersten 300 Bytes (falls vorhanden) der betreffenden VARCHAR- bzw. LONG VARCHAR-Spalte, und :ADDRESSLIND ist das Längen-Indikatorfeld, das die Anzahl der tatsächlich von der Datenbank gelesenen Zeichen enthält.

Wenn sie mit einer sendenden *host-variable* (Ausgangsfeld) verwendet wird, gibt das Längen-Indikatorfeld an, wieviele Zeichen des Ausgangsfeldes an die Datenbank übergeben werden sollen.

Beispiel:

```
DEFINE DATA LOCAL
1 NAMELIND (I2)
1 NAME (A20)
1 AGE (I2)
END-DEFINE
MOVE 4 TO NAMELIND
MOVE 'ABC%' TO NAME
SELECT AGE
INTO :AGE
WHERE NAME LIKE :NAME LINDICATOR :NAMELIND
...
```

Eine LINDICATOR *host-variable* hat Format/Länge I2. Um Verarbeitungszeit zu sparen, sollte sie unmittelbar vor dem betreffenden Ausgangs- bzw. Zielfeld angegeben werden; andernfalls würde sie zur Laufzeit in einen Zwischenspeicher kopiert.

Wenn das LINDICATOR-Feld als I2-Feld definiert ist, wird der SQL-Datentyp VARCHAR zum Senden/Erhalten der betreffenden Spalte verwendet. Wird die LINDICATOR *host-variable* als I4 angegeben, wird ein großer Objektdatentyp (CLOB/BLOB) verwendet.

Wenn das Feld als DYNAMIC (dynamisch) definiert wird, wird die Spalte in einer internen Schleife bis zu ihrer wirklichen Länge gelesen. Das LINDICATOR-Feld und *LENGTH werden auf diese Länge gesetzt. Bei Feldern fester Länge wird die Spalte bis zur definierten Länge gelesen. In beiden Fällen wird das Feld bis zum im LINDICATOR-Feld definierten Wert geschrieben.

Ein Feld fester Länge soll z.B. mit einem als I2 angegebenen LINDICATOR-Feld definiert werden. Wenn die VARCHAR-Spalte mehr Zeichen enthält als in dieses Feld fester Länge passen, wird das Längenindikatorfeld auf die tatsächlich zurückgegebene Länge gesetzt, und das Nullindikatorfeld (falls angegeben) wird auf die Gesamtlänge dieser Spalte (Lesen) gesetzt. Dies ist bei Feldern fester Länge >= 32 KB nicht möglich (die Länge ist größer gewählt als die Länge des Nullindikatorfeldes).

Das Natural-View-Konzept

Einige Natural-SQL-Statements erlauben auch die Verwendung von Natural-Views.

Ein Natural-View kann anstelle einer Parameterliste angegeben werden, wobei jedes Feld des Views — außer Gruppen, redefinierten Feldern sowie Feldern mit vorangestelltem Präfix L@ oder N@ — einem Parameter (*host-variable*) entspricht.

Felder, deren Namen mit “L@” bzw. “N@” anfangen, können nur zusammen mit entsprechenden Feldern gleichen Namens verwendet werden. Dabei werden:

- L@-Felder in LINDICATOR-Felder umgesetzt,
- N@-Felder in INDICATOR-Felder umgesetzt.

Ein L@-Feld sollte im View jeweils unmittelbar vor dem Feld, auf das es sich bezieht, definiert werden.

Beispiel:

```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
  02 PERSID      (I4)
  02 NAME        (A20)
  02 N@NAME      (I2)          /* null indicator of NAME
  02 L@ADDRESS   (I2)          /* length indicator of ADDRESS
  02 ADDRESS     (A50/1:6)
  02 N@ADDRESS   (I2)          /* null indicator of ADDRESS
01 #PERSID      (I4)
END-DEFINE
...
SELECT *
  INTO VIEW PERS
  FROM SQL-PERSONNEL
  WHERE PERSID = #PERSID
...
END-SELECT
```

Das obige Beispiel entspricht dem folgenden:

```
...
SELECT *
  INTO PERSID,
  NAME INDICATOR N@NAME,
  ADDRESS(*)INDICATOR N@ADDRESS LINDICATOR L@ADDRESS
  FROM SQL-PERSONNEL
  WHERE PERSID = #PERSID
...
END-SELECT
```

Skalar-Ausdrücke (*scalar-expressions*)

Eine *scalar-expression* besteht aus einem *factor* und anderen *scalar-expressions* einschließlich Skalar-Operatoren.

$$\left\{ \begin{array}{c} \left[\begin{array}{c} + \\ - \end{array} \right] \left\{ \begin{array}{c} \text{factor} \\ \text{(scalar-expression)} \end{array} \right\} \\ \text{scalar-expression scalar-operator scalar-expression} \end{array} \right\}$$

In punkto Referenzierungspriorität gilt folgendes: Wenn in einer *scalar-expression* ein unqualifizierter Variablenname angegeben wird, wird zunächst angenommen, daß es sich um den Namen einer Spalte der referenzierten Tabelle handelt; falls in der Tabelle eine Spalte dieses Namens nicht vorkommt, behandelt Natural die Variable als Benutzervariable (*host-variable*).

scalar-operator

$$\left\{ \begin{array}{c} + \\ - \\ * \\ / \\ \text{CONCAT} \end{array} \right\}$$

Ein *scalar-operator* kann einer der oben aufgeführten Operatoren sein, wobei vor und nach den Operatoren “-“ und “/” jeweils mindestens ein Leerzeichen stehen muß.

factor

$$\left\{ \begin{array}{c} \text{atom} \\ \text{column-reference} \\ \text{aggregate-function} \\ \text{special-register} \\ \text{scalar-function (scalar-expression, ...)} \\ \text{scalar-expression unit} \\ \text{case-expression} \end{array} \right\}$$

Ein *factor* kann eines der obigen Elemente sein.

atom
$$\left\{ \begin{array}{l} \textit{parameter} \\ \textit{constant} \end{array} \right\}$$

Ein *atom* kann entweder ein *parameter* oder eine Konstante (*constant*) sein; vgl. Abschnitt **Grundlegende Syntaxbestandteile** (Seite 679).

column-reference
$$\left[\begin{array}{l} \textit{table-name} \\ \textit{correlation-name} \end{array} \right] \textit{column-name}$$

Eine *column-reference* ist ein Spaltenname (*column-name*), optional qualifiziert durch einen Tabellennamen (*table-name*) oder einen *correlation-name* (vgl. Abschnitt **Grundlegende Syntaxbestandteile**, Seite 679). Qualifizierte Namen sind oft klarer als unqualifizierte und manchmal erforderlich.

Anmerkung:

Ein “*table-name*” darf hier nicht explizit mit einem “*authorization-identifier*” qualifiziert werden. Falls Sie einen qualifizierten “*table-name*” benötigen, verwenden Sie stattdessen einen “*correlation-name*”.

Wird eine Spalte mit einem *table-name* oder *correlation-name* referenziert, muß sie in der betreffenden Tabelle enthalten sein. Wird weder *table-name* noch *correlation-name* angegeben, muß die betreffende Spalte in einer der in der FROM-Klausel angegebenen Tabellen enthalten sein.

aggregate-function

$$\left\{ \begin{array}{l} \text{COUNT} \left\{ \begin{array}{l} (*) \\ (\text{DISTINCT } \textit{column-reference}) \end{array} \right\} \\ \text{AVG} \\ \text{MAX} \\ \text{MIN} \\ \text{SUM} \end{array} \right\} \left\{ \begin{array}{l} (\text{DISTINCT } \textit{column-reference}) \\ ([\text{ALL}] \textit{scalar-expression}) \end{array} \right\}$$

SQL bietet eine Reihe spezieller Funktionen zur Erweiterung der grundlegenden Suchmöglichkeiten. Folgende sogenannte “SQL aggregate functions” sind verfügbar und werden von Natural unterstützt:

- **AVG** gibt den Durchschnitt der Werte einer Spalte zurück.
- **COUNT** gibt die Anzahl der Werte einer Spalte zurück.
- **MAX** gibt den größten Wert einer Spalte zurück.
- **MIN** gibt den kleinsten Wert einer Spalte zurück.
- **SUM** gibt die Summe der Werte einer Spalte zurück.

Bis auf COUNT(*) sammelt jede dieser Funktionen die Skalarwerte in einem Argument, d.h. einer einzelnen Spalte oder einer *scalar-expression*, und gibt als Ergebnis einen Skalarwert zurück.

Beispiel:

```

DEFINE DATA LOCAL
1 AVGAGE (I2)
END-DEFINE
...
SELECT AVG (AGE)
  INTO AVGAGE
  FROM SQL-PERSONNEL
  ...

```

Im allgemeinen kann dem Argument optional das Schlüsselwort **DISTINCT** vorangestellt werden, um doppelte Werte zu eliminieren, bevor die Funktion ausgeführt wird.

Wenn Sie **DISTINCT** angeben, muß das Argument der Name einer einzelnen Spalte sein; wenn Sie **DISTINCT** nicht angeben, kann das Argument eine allgemeine *scalar-expression* sein.

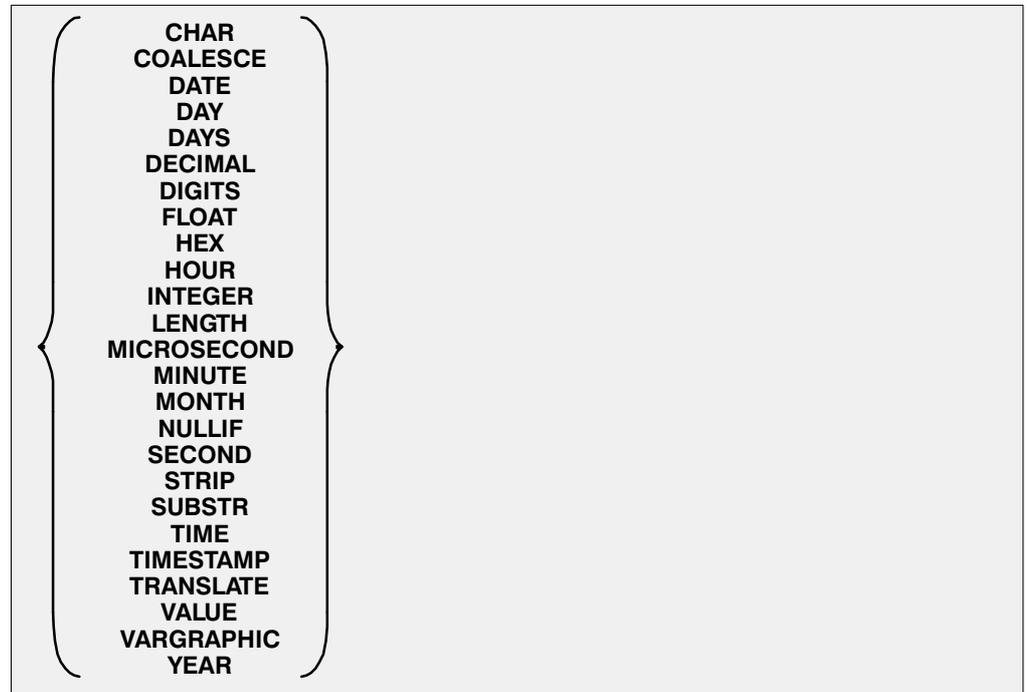
DISTINCT ist nicht erlaubt mit der Funktion **COUNT(*)**, mit der alle Reihen in einer Tabelle — ohne Eliminierung doppelt vorkommender Reihen — gezählt werden.

special-register

USER
CURRENT TIMEZONE
CURRENT DATE
CURRENT TIME
CURRENT TIMESTAMP
CURRENT SQLID
CURRENT PACKAGESET
CURRENT SERVER

Bei der Referenzierung eines *special-registers* erhält man einen Skalarwert.

Special-register sind — mit Ausnahme von **USER** — nicht Teil der Standard-SQL und werden daher nur als Teil des Natural SQL Extended Set unterstützt.

scalar-function

CHAR
COALESCE
DATE
DAY
DAYS
DECIMAL
DIGITS
FLOAT
HEX
HOURL
INTEGER
LENGTH
MICROSECOND
MINUTE
MONTH
NULLIF
SECOND
STRIP
SUBSTR
TIME
TIMESTAMP
TRANSLATE
VALUE
VARGRAPHIC
YEAR

Scalar-functions sind eingebaute, bei der Konstruktion von Skalar-Berechnungen verwendbare Funktionen. Sie sind nicht Teil der Standard-SQL. Obige *scalar-functions* werden als Teil des Natural SQL Extended Set unterstützt.

units

{ YEAR YEARS MONTH MONTHS DAY DAYS HOUR HOURS MINUTE MINUTES SECOND SECONDS MICROSECOND MICROSECONDS }

Units sind nicht Teil der Standard-SQL und werden daher nur als Teil des Natural SQL Extended Set unterstützt.

case-expression

CASE { <i>searched-when-clause...</i> <i>simple-when-clause</i> } [ELSE { NULL <i>scalar-expression</i> }] END
--

Case-expressions sind nicht Teil der Standard-SQL und werden daher nur als Teil des Natural SQL Extended Set unterstützt.

searched-when-clause

WHEN <i>search-condition</i> THEN { NULL <i>scalar-expression</i> }

Näheres zu *search-condition* siehe Seite 698.

simple-when-clause

<i>scalar-expression</i> { WHEN <i>scalar-expression</i> THEN { NULL <i>scalar-expression</i> } } ...

Suchbedingungen (*search-conditions*)

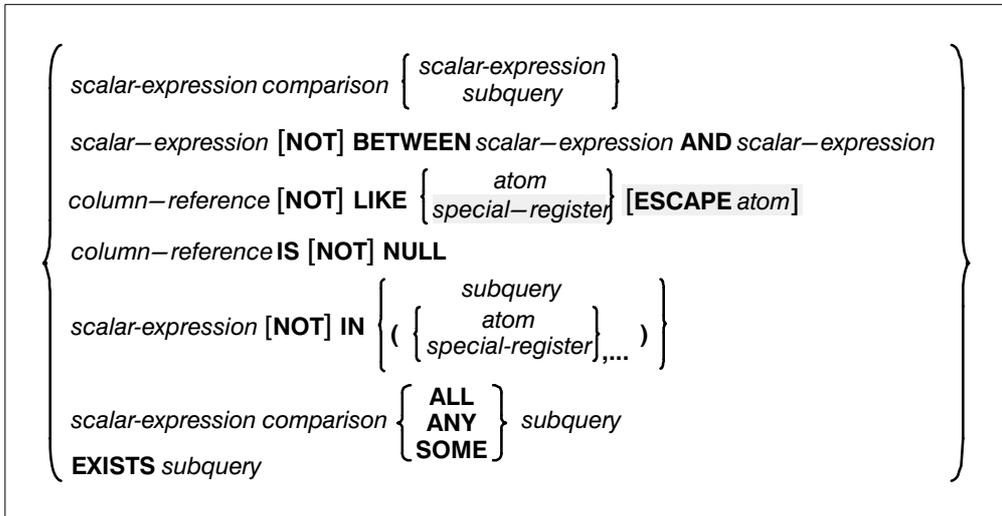
search-condition

$$\left\{ \begin{array}{l} [\text{NOT}] \left\{ \begin{array}{l} \textit{predicate} \\ (\textit{search-condition}) \end{array} \right\} \\ \textit{search-condition} \left\{ \begin{array}{l} \text{AND} \\ \text{OR} \end{array} \right\} \textit{search-condition} \end{array} \right\}$$

Eine *search-condition* kann aus einer einfachen Bedingung (*predicate*) bestehen oder aus mehreren *search-conditions*, die durch die Boole'schen Operatoren AND, OR und NOT verknüpft werden, wobei die Reihenfolge der Auswertung außerdem durch entsprechende Klammerung bestimmt werden kann.

Beispiel:

```
DEFINE DATA LOCAL
01 NAME    (A20)
01 AGE     (I2)
END-DEFINE
...
SELECT *
  INTO NAME, AGE
  FROM SQL-PERSONNEL
  WHERE AGE = 32 AND NAME > 'K'
END-SELECT
...
```

predicate

Das *predicate* gibt eine Bedingung an, die “wahr”, “falsch” oder “unbekannt” sein kann. In einer *search-condition* kann ein *predicate* aus einer einfachen oder komplexen Vergleichsoperation oder einer anderen Art von Bedingung bestehen.

Beispiel:

```

SELECT NAME, AGE
  INTO VIEW PERS
  FROM SQL-PERSONNEL
 WHERE AGE BETWEEN 20 AND 30
    OR AGE IN ( 32, 34, 36 )
    AND NAME LIKE '%er'
    ...

```

Anmerkung:

Das Prozentzeichen (%) kann zu Konflikten mit Natural-Terminalkommandos führen. In diesem Fall definieren Sie als Steuerzeichen für Terminalkommandos ein anderes Zeichen als “%” (vgl. Session-Parameter CF im Natural-Referenzhandbuch).

Die einzelnen *predicates* sind auf den folgenden Seiten beschrieben (weitere Informationen zu *predicates* finden Sie in der betreffenden Literatur). Entsprechend der obigen Syntax heißen sie wie folgt:

- Comparison Predicate
- BETWEEN Predicate
- LIKE Predicate
- NULL Predicate
- IN Predicate
- Quantified Predicate
- EXISTS Predicate.

Comparison Predicate

$$\text{scalar-expression comparison } \left\{ \begin{array}{l} \text{scalar-expression} \\ \text{subquery} \end{array} \right\}$$

Ein Comparison Predicate vergleicht zwei Werte.

Näheres zu *scalar-expressions* siehe Seite 692.

comparison

$$\left\{ \begin{array}{l} = \\ < \\ > \\ <= \\ >= \\ <> \\ \lrcorner = \\ \lrcorner > \\ \lrcorner < \end{array} \right\}$$

Comparison kann einer der folgenden Operatoren sein:

=	gleich
<	kleiner als
>	größer als
<=	kleiner gleich
>=	größer gleich
<>	ungleich
⌊ =	ungleich
⌊ >	nicht größer als
⌊ <	nicht kleiner als

subquery

(*select-expression*)

Eine *subquery* ist eine *select-expression* innerhalb einer anderen *select-expression*.

Beispiel:

```
DEFINE DATA LOCAL
1 #NAME      (A20)
1 #PERSNR    (I4)
END-DEFINE
...
SELECT NAME, PERSNR
  INTO #NAME, #PERSNR
  FROM SQL-PERSONNEL
  WHERE PERSNR IN
    ( SELECT PERSNR
      FROM SQL-AUTOMOBILES
      WHERE COLOR = 'black' )
  ...
END-SELECT
```

Näheres zu *select-expression* siehe Seite 705.

BETWEEN Predicate

```
scalar-expression [NOT] BETWEEN scalar-expression AND scalar-expression
```

Ein BETWEEN Predicate vergleicht einen Wert mit einem Bereich von Werten.

Näheres zu *scalar-expression* siehe Seite 692.

LIKE Predicate

```
column-reference [NOT] LIKE { atom  
 special-register } [ESCAPE atom]
```

Ein LIKE Predicate sucht nach Zeichenketten, die ein bestimmtes Muster haben.

Näheres zu *column-reference*, *atom* und *special-register* finden Sie unter **Skalar-Ausdrücke** (Seite 692).

NULL Predicate

```
column-reference IS [NOT] NULL
```

Ein NULL Predicate prüft auf Nullwerte.

Näheres zu *column-reference* siehe Seite 693.

IN Predicate

```
scalar-expression [NOT] IN { ( { subquery  
 atom  
 special-register }, ... ) }
```

Ein IN Predicate vergleicht einen Wert mit einer Sammlung von Werten.

Näheres zu *scalar-expression*, *atom* und *special-register* finden Sie unter **Skalar-Ausdrücke** (Seite 692).

Näheres zu *subquery* siehe Seite 702.

Quantified Predicate

$$\text{scalar-expression comparison} \left\{ \begin{array}{l} \text{ALL} \\ \text{ANY} \\ \text{SOME} \end{array} \right\} \text{subquery}$$

Ein Quantified Predicate vergleicht einen Wert mit einer Sammlung von Werten.

Näheres zu *scalar-expression* siehe Seite 692, zu *comparison* siehe Seite 701, zu *subquery* siehe Seite 702.

EXISTS Predicate

EXISTS *subquery*

Ein EXISTS Predicate prüft, ob bestimmte Reihen vorhanden sind.

Die Bedingung des EXISTS Predicate kann nur erfüllt werden, wenn die ausgewertete *subquery* tatsächlich ein Ergebnis liefert, d.h. wenn mindestens eine Reihe in der FROM-Tabelle der *subquery* die WHERE-Bedingung dieser *subquery* erfüllt.

Beispiel für EXISTS:

```

DEFINE DATA LOCAL
1 #NAME      (A20)
END-DEFINE
...
SELECT NAME
  INTO #NAME
  FROM SQL-PERSONNEL
  WHERE EXISTS
    ( SELECT *
      FROM SQL-EMPLOYEES
      WHERE PERSNR > 1000
        AND NAME < 'L' )
    ...
END-SELECT
...

```

Näheres zu *subquery* siehe Seite 702.

Selektionsausdrücke (*select-expressions*)

select-expression

```
SELECT selection table-expression
```

Eine *select-expression* gibt eine Ergebnistabelle an.

selection

```
[ ALL
DISTINCT ] { { scalar-expression [ AS correlation-name ] } , .. }
*
```

In der *selection* geben Sie an, was ausgewählt werden soll.

ALL/DISTINCT

Doppelt vorkommende Reihen werden nicht automatisch aus dem Ergebnis einer *select-expression* entfernt. Wenn Sie dies wünschen, geben Sie das Schlüsselwort **DISTINCT** an.

Die Alternative zu **DISTINCT** ist **ALL**. Wenn Sie nichts angeben, gilt **ALL**.

scalar-expression

Anstelle von oder zusätzlich zu einfachen Spaltennamen können Sie auch allgemeine Skalar-Ausdrücke (*scalar-expression*) angeben, die Skalar-Operatoren und Skalar-Funktionen, die berechnete Werte liefern, enthalten (vgl. Abschnitt **Skalar-Ausdrücke**, Seite 692).

Beispiel:

```
SELECT NAME, 65 - AGE
FROM SQL-PERSONNEL
...
```

correlation-name

Es besteht die Möglichkeit, einer *scalar-expression* einen *correlation-name* als Alias-Namen für eine Ergebnisspalte zuzuweisen.

Der *correlation-name* braucht nicht eindeutig sein. Wenn für eine Ergebnisspalte kein *correlation-name* angegeben wird, wird der betreffende *column-name* genommen (falls sich die Ergebnisspalte von einem Spaltennamen ableitet; andernfalls hat die Ergebnisspalte keinen Namen). Der Name einer Ergebnisspalte kann beispielsweise als Spaltenname in der ORDER BY-Klausel eines SELECT-Statements angegeben werden.

Stern-Notation (*)

Alle Spalten aller in der FROM-Klausel angegebenen Tabellen werden ausgewählt.

Beispiel:

```
SELECT *  
  FROM SQL-PERSONNEL, SQL-AUTOMOBILES  
  ...
```

table-expression

```
FROM table-reference,...  
    [WHERE search-condition]  
    [GROUP BY column-reference,...]  
    [HAVING search-condition]
```

Die *table-expression* gibt an, von wo und nach welchen Kriterien Reihen gelesen werden sollen.

FROM-Klausel

```
FROM table-reference,...
```

table-reference

```
{  
  table-name [[AS] correlation-name]  
  subquery [AS] correlation-name  
  joined-table  
}
```

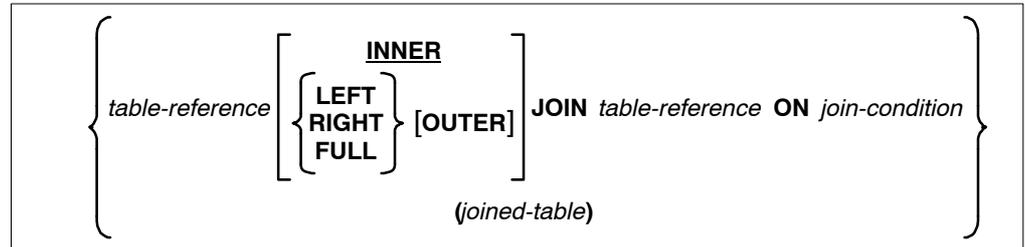
In der FROM-Klausel geben Sie eine oder mehrere Tabellen an, die die in der *selection-list* verwendeten Spaltenfelder enthalten müssen.

Sie können entweder eine einzelne Tabelle angeben oder eine Zwischenergebnistabelle erzeugen, die das Ergebnis einer *subquery* oder einer "Join"-Operation ist (siehe unten).

Da in einer FROM-Klausel mehrere Tabellen (d.h. DDMs) angesprochen werden können und eine *table-expression* mehrere FROM-Klauseln enthalten kann, wenn *subqueries* angegeben sind, bestimmt die Datenbanknummer (DBID) des ersten DDMs in der ersten FROM-Klausel die Datenbank.

Es besteht die Möglichkeit, einem *table-name* einen *correlation-name* zuzuweisen. Bei einer *subquery* muß ein *correlation-name* zugewiesen werden.

Näheres zu *table-name* siehe Seite 680, zu *correlation-name* siehe Seite 683 und zu *subquery* siehe Seite 702.

joined-table

Eine *joined-table* gibt eine Zwischenergebnistabelle an, die das Ergebnis einer “Join”-Operation ist.

Der “Join” kann ein INNER, LEFT OUTER, RIGHT OUTER oder FULL OUTER JOIN sein. Falls Sie nichts angeben, gilt “INNER”.

Es ist möglich, mehrere “Joins” zu schachteln, d.h. die Tabellen, die die Zwischenergebnistabelle bilden, können ihrerseits Zwischenergebnistabellen einer JOIN-Operation oder einer *subquery* sein, wobei letztere wiederum ebenfalls eine *joined-table* oder eine weitere *subquery* in der FROM-Klausel haben kann.

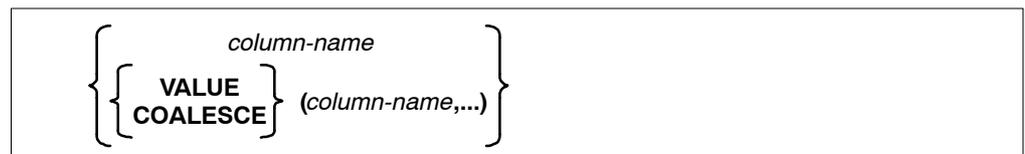
join-condition

$\text{join-expression comparison join-expression [AND join-condition] ...}$
--

Mehrere *join-conditions* können mittels AND miteinander verknüpft werden.

Bei einem FULL OUTER JOIN ist nur das Gleichheitszeichen (=) als *comparison* erlaubt. Näheres zu *comparison* siehe Seite 701.

Die erste *join-expression* muß sich auf die erste *table-reference*, die zweite *join-expression* auf die zweite *table-reference* beziehen.

join-expression

In einer *join-expression* sind nur *column-names* und die *scalar-function* VALUE (bzw. ihr Synonym COALESCE) erlaubt. Näheres zu *column-name* siehe Seite 680.

WHERE-Klausel

```
[WHERE search-condition]
```

In der WHERE-Klausel geben Sie eine Suchbedingung (*search-condition*) an, nach der die Reihen gelesen werden sollen.

Beispiel:

```
DEFINE DATA LOCAL
01 NAME      (A20)
01 AGE       (I2)
END-DEFINE
...
SELECT *
  INTO NAME, AGE
  FROM SQL-PERSONNEL
  WHERE AGE = 32
END-SELECT
...
```

Näheres zu *search-condition* siehe Seite 698.

GROUP BY-Klausel

[**GROUP BY** *column-reference*,...]

Die GROUP BY-Klausel sortiert die in der FROM-Klausel angegebene Tabelle nach Gruppen, und zwar so, daß alle Reihen einer Gruppe in der GROUP BY-Spalte den gleichen Wert haben.

Jede *column-reference* in der Selektionsliste muß entweder eine GROUP BY-Spalte sein oder mit einer *aggregate-function* angegeben werden. *Aggregate-functions* werden auf einzelne Gruppen (nicht auf die ganze Tabelle) angewandt. Die Ergebnistabelle enthält soviele Reihen wie Gruppen.

Näheres zu *column-reference* und *aggregate-function* siehe Seite 693 bzw. Seite 694.

Beispiel:

```
DEFINE DATA LOCAL
1 #AGE      (I2)
1 #NUMBER  (I2)
END-DEFINE
...
SELECT AGE , COUNT(*)
      INTO #AGE, #NUMBER
      FROM SQL-PERSONNEL
      GROUP BY AGE
...

```

Steht vor der GROUP BY-Klausel eine WHERE-Klausel, werden nur diejenigen Reihen von der GROUP BY-Klausel erfaßt, die die WHERE-Bedingung erfüllen.

HAVING-Klausel

[**HAVING** *search-condition*]

Wenn Sie eine HAVING-Klausel verwenden, sollten Sie auch eine GROUP BY-Klausel verwenden.

Genau wie die WHERE-Klausel Reihen aus einer Ergebnistabelle aussortiert, sortiert die HAVING-Klausel Gruppen aus, und zwar auf Grundlage eines Suchkriteriums (*search-criterion*). *Scalar-expressions* in einer HAVING-Klausel dürfen pro Gruppe nur einen Wert enthalten.

Näheres zu *scalar-expression* und *search-condition* siehe Seite 692 bzw. Seite 698.

Beispiel:

```
DEFINE DATA LOCAL
1 #NAME      (A20)
1 #AVGAGE    (I2)
1 #NUMBER    (I2)
END-DEFINE
...
SELECT NAME, AVG(AGE), COUNT(*)
INTO #NAME, #AVGAGE, #NUMBER
FROM SQL-PERSONNEL
GROUP BY NAME
HAVING COUNT(*) > 1
...
```

Flexible SQL

Zusätzlich zu der im bisherigen Verlauf dieses Kapitels beschriebenen SQL-Syntax haben Sie mit *flexibler SQL* die Möglichkeit, beliebige SQL-Syntax zu verwenden.

Die Zeichen “<<” und “>>”

Flexible SQL muß zwischen den Zeichen “<<” und “>>” stehen. Sie kann beliebigen SQL-Text und *host-variables* enthalten. Mit flexibler SQL verwendete *host-variables* müssen als Präfix einen Doppelpunkt (:) haben.

Flexible SQL kann über mehrere Zeilen gehen und kann ganze oder teilweise Kommentarzeilen enthalten (vgl. PROCESS SQL-Statement, Seite 727).

Flexible SQL kann anstelle folgender SQL-Syntaxteile verwendet werden:

- *atom*
- *column-reference*
- *scalar-expression*
- *predicate*

Flexible SQL kann auch zwischen den Klauseln einer *select-expression* verwendet werden:

```
SELECT selection
  << ... >>
  INTO ...
  FROM ...
  << ... >>
  WHERE ...
  << ... >>
  GROUP BY ...
  << ... >>
  HAVING ...
  << ... >>
  ORDER BY ...
  << ... >>
```

Anmerkung:

Der in flexibler SQL angegebene SQL-Text wird nicht vom Natural-Compiler verarbeitet, sondern (mit ausgetauschten host-variables) einfach in die SQL-Zeichenkette kopiert, die an das Datenbanksystem übergeben wird. Demzufolge werden Syntaxfehler in der flexiblen SQL erst zur Laufzeit erkannt, wenn die Datenbank das betreffende Statement ausführt.

Beispiel 1:

```
SELECT NAME
FROM SQL-EMPLOYEES
WHERE << MONTH (BIRTH) >> = << MONTH (CURRENT_DATE) >>
```

Beispiel 2:

```
SELECT NAME
FROM SQL-EMPLOYEES
WHERE << MONTH (BIRTH) = MONTH (CURRENT_DATE) >>
```

Beispiel 3:

```
SELECT NAME
FROM SQL-EMPLOYEES
WHERE SALARY > 50000
<< INTERSECT
  SELECT NAME
  FROM SQL-EMPLOYEES
  WHERE DEPT = 'DEPT10'
>>
```

Textvariablen

```
<< :T:host-variable [LINDICATOR :host-variable] >>
```

Innerhalb der flexiblen SQL können Sie auch sogenannte “Textvariablen” angeben.

:T:

Eine Textvariable ist eine *host-variable* mit dem Präfix “:T:”. Sie muß alphanumerisches Format haben.

Zur Laufzeit wird eine Textvariable innerhalb eines SQL-Statements durch ihren Inhalt ersetzt, d.h. die in der Textvariablen enthaltene Textzeichenkette wird in die SQL-Zeichenkette eingefügt.

Nach dem Ersetzen werden nachfolgende Leerzeichen aus der eingefügten Textzeichenkette entfernt.

Sie müssen selbst darauf achten, daß sich aus dem Inhalt einer Textvariablen beim Einfügen ein syntaktisch korrektes SQL-Statement ergibt. Insbesondere darf eine Textvariable keine *host-variables* enthalten.

Ein Statement, das eine Textvariable enthält, wird immer im dynamischen SQL-Modus ausgeführt.

LINDICATOR-Option

Nach der Textvariablen können Sie das Schlüsselwort LINDICATOR sowie eine Längenindikator-Variable (d.h. eine *host-variable* mit vorangestelltem Doppelpunkt) angeben.

Die Längenindikator-Variable muß Format/Länge I2 haben.

Wenn Sie keine LINDICATOR-Variable angeben, wird der gesamte Inhalt der Textvariablen in die SQL-Zeichenkette eingefügt.

Wenn Sie eine LINDICATOR-Variable angeben, werden nur die ersten *n* Zeichen (wobei *n* der Wert der LINDICATOR-Variablen ist) des Textvariableninhalts in die SQL-Zeichenkette eingefügt. Falls die Zahl in der LINDICATOR-Variablen größer als die Länge des Textvariableninhalts ist, wird der gesamte Textvariableninhalt eingefügt. Falls die Zahl in der LINDICATOR-Variablen negativ oder 0 ist, wird nichts eingefügt.

Allgemeine Informationen zu *host-variables* siehe Seite 684.

Beispiel mit Textvariable:

```
DEFINE DATA LOCAL
01 TEXTVAR (A200)
01 TABLES VIEW OF SYSIBM-SYSTABLES
    02 NAME
    02 CREATOR
END-DEFINE
*
MOVE 'WHERE NAME > ''SYS'' AND CREATOR = ''SYSIBM'' ' TO TEXTVAR
*
SELECT * INTO VIEW TABLES
    FROM SYSIBM-SYSTABLES
    << :T:TEXTVAR >>
    DISPLAY TABLES
END-SELECT
*
END
```

Das generierte SQL-Statement (wie mit dem Systemkommando LISTSQL angezeigt) sieht wie folgt aus:

```
SELECT NAME, CREATOR FROM SYSIBM.SYSTABLES:T: FOR FETCH ONLY
```

Das ausgeführte SQL-Statement sieht wie folgt aus:

```
SELECT TABNAME, CREATOR FROM SYSIBM.SYSTABLES
    WHERE TABNAME > 'SYS' AND CREATOR = 'SYSIBM'
```

Statements

CALLDBPROC

```

CALLDBPROC dbproc ddm-name
  [USING] [ parameter [ AD= { M }
  { O }
  { A } ] ] ...
  [RESULT SETS result-set...]
  [GIVING sqlcode]
  [CALLMODE= { NONE }
  { NATURAL } ]

```

Funktion

Das Statement CALLDBPROC dient dazu, eine “Stored Procedure” des SQL-Datenbanksystems, mit dem Natural verbunden ist, aufzurufen.

Die Stored Procedure kann entweder ein Natural-Subprogramm oder ein in einer anderen Programmiersprache geschriebenes Programm sein.

Neben der Möglichkeit, Parameter zwischen dem aufrufenden Objekt und der Stored Procedure zu übergeben, unterstützt CALLDBPROC sogenannte “Result Sets”; mit diesen ist es möglich, größere Datenmengen von der Stored Procedure an das aufrufende Objekt zu übergeben als dies mittels Parametern möglich wäre.

Die Result Sets sind von der Stored Procedure erzeugte “temporäre Ergebnistabellen”, die das aufrufende Objekt mittels eines READ RESULT SET-Statements lesen und verarbeiten können.

Anmerkung:

Im Prinzip ist der Aufruf einer Stored Procedure mit dem Aufruf eines Natural-Subprogramms vergleichbar: Wenn das CALLDBPROC-Statement ausgeführt wird, wird die Kontrolle an die Stored Procedure übergeben; nach Verarbeitung der Stored Procedure wird die Kontrolle wieder an das aufrufende Objekt zurückgegeben, und die Verarbeitung wird mit dem nächsten Statement nach dem CALLDBPROC-Statement fortgesetzt.

dbproc

Als *dbproc* geben Sie den Namen der Stored Procedure an, die aufgerufen werden soll. Der Name kann entweder als alphanumerische Variable oder als Konstante (in Apostrophen) angegeben werden.

Der Name muß den Regeln für Stored-Procedure-Namen des Ziel-Datenbanksystems entsprechen.

Falls die Stored Procedure ein Natural-Subprogramm ist, darf der eigentliche Procedure-Name nicht länger als 8 Stellen sein.

ddm-name

Der Name eines DDMs muß angegeben werden, um die “Adresse” der Datenbank, die die Stored Procedure ausführt, bereitzustellen. Weitere Informationen zu *ddm-name* siehe Seite 680.

parameter

Als *parameter* können Sie Parameter angeben, die vom aufrufenden Objekt an die Stored Procedure übergeben werden sollen.

Ein *parameter* kann eine *host-variable* (siehe Seite 684, optional mit INDICATOR- und LINDICATOR-Klauseln), eine *constant* (siehe Seite 679) oder das Schlüsselwort NULL sein.

AD=

Wenn es sich bei dem *parameter* um eine *host-variable* handelt, können Sie sie als nicht-modifizierbar (AD=O), modifizierbar (AD=M) oder nur für Eingabe (AD=A) markieren.

- **AD=M** — Standardmäßig kann der übergebene Wert eines Parameters in der Stored Procedure geändert werden und der geänderte Werte an das aufrufende Objekt zurückgegeben werden, wo er den ursprünglichen Wert überschreibt. Dies ist die Standardeinstellung. (Entsprechende Procedure-Notation in DB2 für OS/390: INOUT.)
- **AD=O** — Wenn Sie einen Parameter mit AD=O markieren, kann der übergebene Wert in der Stored Procedure geändert werden, aber der geänderte Wert kann nicht an das aufrufende Objekt zurückgegeben werden; d. h. das Feld im aufrufenden Objekt behält seinen ursprünglichen Wert. (Entsprechende Procedure-Notation in DB2 für OS/390: IN.)
- **AD=A** — Wenn Sie einen Parameter mit AD=A markieren, wird sein Wert nicht *an* die Stored Procedure übergeben, aber er erhält einen Wert *von* der Stored Procedure. (Entsprechende Procedure-Notation in DB2 für OS/390: OUT.)

Wenn der *parameter* eine *constant* ist, kann AD nicht explizit angegeben werden. Für Konstanten gilt immer AD=O.

result-set

Als *result-set* geben Sie ein Feld an, in das der Result-Set-Locator zurückgegeben werden soll.

Ein *result-set* muß eine Variable von Format/Länge I4 sein.

Der Wert einer *result-set*-Variablen ist lediglich eine Zahl, die den Result Set identifiziert und die in einem nachfolgenden READ RESULT SET-Statement referenziert werden kann.

Die Reihenfolge der *result-set*-Werte entspricht der Reihenfolge der von der Stored Procedure zurückgegebenen Result Sets.

Der Inhalt der Result Sets kann von einem nachfolgenden READ RESULT SET-Statement verarbeitet werden.

Wenn kein Result Set zurückgegeben wird, enthält die betreffende *result-set*-Variable "0".

Auf Großrechnern können mehrere Result Sets angegeben werden. Auf allen anderen Plattformen kann nur ein Result Set angegeben werden.

GIVING *sqlcode*

Mit dieser Option können Sie den SQL-Code des SQL CALL-Statements erhalten, das die Stored Procedure aufruft.

Wenn Sie diese Option angeben und der SQL-Code der Stored Procedure ist nicht "0", wird keine Natural-Fehlermeldung ausgegeben. In diesem Fall muß die als Reaktion auf den SQL-Codewert auszuführende Handlung im aufrufenden Natural-Objekt programmiert werden.

Das *sqlcode*-Feld muß eine Variable von Format/Länge I4 sein.

Wenn Sie die Option GIVING *sqlcode* nicht verwenden, gibt Natural eine Fehlermeldung aus, falls der SQL-Code der Stored Procedure nicht "0" ist.

CALLMODE

Ist die Stored Procedure ein Natural-Subprogramm, müssen Sie CALLMODE=NATURAL angeben.

Anmerkung:

CALLMODE=NATURAL wirkt sich auch auf interne Parameter aus, die an die bzw. von der Stored Procedure übergeben werden; Näheres hierzu finden Sie im jeweiligen Natural-Datenbankschnittstellen-Handbuch.

CALLMODE=NATURAL steht nur auf Großrechnern zur Verfügung.

Beispiel

Das folgende Beispiel zeigt ein Natural-Programm, das die Stored Procedure 'demo_proc' aufruft, um alle zu einem gegebenen Bereich gehörenden Namen der Tabelle PERSON einzulesen.

Drei Parameter-Felder werden an 'demo_proc' übergeben: der erste und zweite Parameter übergeben jeweils Start- und Endwerte des Bereichs von Namen an die Stored Procedure, und der dritte Parameter nimmt einen Namen auf, der das Kriterium erfüllt.

In diesem Beispiel werden die Namen in einem Result Set zurückgegeben, der mit dem READ RESULT SET-Statement verarbeitet wird.

```

DEFINE DATA LOCAL
1 PERSON VIEW OF DEMO-PERSON
  2 PERSON_ID
  2 LAST_NAME
1 #BEGIN (A2) INIT <'AB'>
1 #END      (A2) INIT <'DE'>
1 #RESPONSE (I4)
1 #RESULT   (I4)
1 #NAME     (A20)
END-DEFINE

...

CALLDBPROC 'demo_proc' DEMO-PERSON #BEGIN (AD=0) #END (AD=0) #NAME (AD=A)
RESULT SETS #RESULT
GIVING #RESPONSE

READ RESULT SET #RESULT INTO #NAME FROM DEMO-PERSON
GIVING #RESPONSE
DISPLAY #NAME
END-RESULT

...

END

```

Weitere Beispiele siehe entsprechendes Natural-Datenbankschnittstellen-Handbuch in der Dokumentation für Natural für Großrechner.

COMMIT

COMMIT

Funktion

Das SQL-Statement COMMIT entspricht dem END TRANSACTION-Statement. Es markiert das Ende einer logischen Transaktion und bewirkt, daß alle während der Transaktion gesperrten Daten freigegeben werden. Alle Datenänderungen werden bestätigt und auf der Datenbank physisch durchgeführt.

Beispiel:

```
...  
DELETE FROM SQL-PERSONNEL WHERE NAME = 'SMITH'  
COMMIT  
...
```

Da bei Beendigung einer logischen Arbeitseinheit alle Cursor geschlossen werden, darf ein COMMIT-Statement nicht innerhalb einer datenbankverändernden Verarbeitungsschleife stehen, sondern muß nach einer solchen stehen (bzw. bei geschachtelten Schleifen nach der äußersten Schleife).

Hinweis für Nicht-Natural-Programme

Wenn ein Natural-Programm ein Nicht-Natural-Programm aufruft, sollte das aufgerufene Programm kein eigenes COMMIT-Statement enthalten, falls das aufrufende Natural-Programm selbst auch Datenbankaufrufe durchführt. In diesem Falle sollte das Natural-Programm das COMMIT-Statement für das Nicht-Natural-Programm enthalten.

DELETE

Syntax 1 — Searched DELETE

```
DELETE FROM table-name [correlation-name] [WHERE search-condition]
```

Syntax 2 — Positioned DELETE

```
DELETE FROM table-name WHERE CURRENT OF CURSOR [(r)]
```

Funktion

Das SQL-Statement DELETE dient dazu, Reihen aus einer Tabelle zu löschen, ohne einen Cursor zu verwenden (“searched” DELETE), oder Reihen aus einer Tabelle zu löschen, auf die der Cursor zeigt (“positioned” DELETE).

“Searched DELETE” ist ein eigenständiges Statement, das unabhängig von einem SELECT-Statement verwendet werden kann. Mit einem einzigen Statement können sie keine, eine, mehrere oder alle Reihen einer Tabelle löschen. Welche Reihen gelöscht werden, bestimmen Sie mit einer Suchbedingung (*search-condition*), die auf die Tabelle angewandt wird. Außerdem ist es möglich, dem Tabellennamen einen *correlation-name* zuzuweisen.

Ein “positioned DELETE” bezieht sich auf einen Cursor innerhalb einer Datenbankschleife. Es muß daher dieselbe Tabelle referenzieren wie das entsprechende SELECT-Statement, sonst wird eine Fehlermeldung ausgegeben. Ein “positioned DELETE” kann nur mit cursor-orientierter Selektion verwendet werden. In seiner Funktion entspricht “positioned DELETE” dem “gewöhnlichen” Natural-Statement DELETE.

Anmerkung:

*Die Anzahl der Reihen, die mit einem “searched DELETE” tatsächlich gelöscht wurden, kann mit der Systemvariablen *ROWCOUNT (siehe Natural Referenzhandbuch) ermittelt werden.*

FROM-Klausel

In der FROM-Klausel geben Sie an, aus welcher Tabelle die Reihen gelöscht werden sollen.

WHERE-Klausel

In der WHERE-Klausel geben Sie das Auswahlkriterium an, das bestimmt, welche Reihen gelöscht werden sollen.

Wenn Sie keine WHERE-Klausel angeben, wird die gesamte Tabelle gelöscht.

Statement-Referenzierung (*r*)

Die Notation “(*r*)” dient zur Referenzierung des Statements, das zur Selektion der zu löschenden Reihe verwendet wurde. Wenn keine Statement-Referenz angegeben wird, bezieht sich das DELETE-Statement auf die jeweils innerste aktive Verarbeitungsschleife, mit der der Datensatz, der gelöscht werden soll, ausgewählt wurde.

INSERT

$$\text{INSERT INTO } table\text{-name} \left\{ \begin{array}{l} (*) \text{ VALUES (VIEW } view\text{-name)} \\ [(column\text{-list})] \left\{ \begin{array}{l} \text{VALUES (} \left\{ \begin{array}{l} \text{VIEW } view\text{-name} \\ insert\text{-item-list} \end{array} \right\} \text{)} \\ select\text{-expression} \end{array} \right\} \end{array} \right\}$$

Funktion

Das SQL-Statement INSERT dient dazu, einer Tabelle eine oder mehrere neue Reihen hinzuzufügen.

INTO-Klausel

In der INTO-Klausel geben Sie an, welcher Tabelle Reihen hinzugefügt werden sollen.

Näheres zu *table-name* siehe Seite 680.

column-list

column-name,...

In der *column-list* können Sie eine oder mehrere Spalten angeben, die in der hinzugefügten Reihe Werte erhalten sollen.

Die Reihenfolge der angegebenen Spalten muß der Reihenfolge der Werte entsprechen, die in der *insert-item-list* oder im angegebenen View sind (siehe unten).

Wenn Sie keine *column-list* angeben, werden die in der *insert-item-list* bzw. im View angegebenen Werte entsprechend der impliziten Liste aller Spalten eingefügt, und zwar in der Reihenfolge, in der sie in der Tabelle stehen.

VALUES-Klausel

Mit der VALUES-Klausel fügen Sie *eine einzelne* Reihe in die Tabelle ein. Der VALUES-Klausel kann entweder ein Stern (*) oder eine *column-list* vorangestellt werden, und sie hat dementsprechend eine der folgenden Formen:

VALUES-Klausel mit Stern-Notation

```
VALUES (VIEW view-name)
```

Wenn Sie Stern-Notation angeben, **müssen** Sie in der VALUES-Klausel einen View angeben. Mit den Feldwerten des Views wird dann eine neue Reihe in die Tabelle eingefügt, wobei die Feldnamen des Views als Spaltennamen der Reihe verwendet werden.

VALUES-Klausel mit *column-list*

```
VALUES ( { VIEW view-name }  
         insert-item-list )
```

Wenn Sie eine *column-list* angeben und in der VALUES-Klausel einen View referenzieren, muß die Anzahl der Spalten in der *column-list* der Anzahl der Felder im View entsprechen.

Wenn Sie keine *column-list* angeben, werden die im View angegebenen Werte entsprechend der impliziten Liste aller Spalten in der Reihenfolge, in der sie in der Tabelle stehen, eingefügt.

insert-item-list

In der *insert-item-list* können Sie einen oder mehrere Werte angeben, die den in der *column-list* angegebenen Spalten zugewiesen werden sollen. Die Reihenfolge der angegebenen Werte muß der Reihenfolge der Spalten entsprechen.

Wenn Sie keine *column-list* angeben, werden die in der *insert-item-list* angegebenen Werte entsprechend der impliziten Liste aller Spalten in der Reihenfolge, in der sie in der Tabelle stehen, eingefügt.

Die Werte können Konstanten (*constant*), Benutzervariablen (*parameter*), “*special-registers*” oder NULL sein.

Näheres zu *view-name*, *constant* und *parameter* finden Sie unter **Grundlegende Syntaxbestandteile** (Seite 679), Näheres zu *special-register* finden Sie auf Seite 695.

Wenn der Wert NULL zugewiesen wird, bedeutet dies, daß das betreffende Feld keinen Wert erhält (auch nicht den Wert "0" oder "Leerzeichen").

Beispiel — INSERT einer einzelnen Reihe:

```
...  
INSERT INTO SQL-PERSONNEL (NAME,AGE)  
VALUES ('ADKINSON',35)  
...
```

select-expression

Mit einer *select-expression* fügen Sie *mehrere* Reihen in eine Tabelle ein. Die *select-expression* wird ausgewertet, und jede Reihe der resultierenden Tabelle wird so behandelt, als ob die Werte der Reihe als Werte in einer VALUES-Klausel einer einreihigen INSERT-Operation angegeben worden wären.

Näheres zu *selection-expression* siehe Seite 705.

Beispiel — INSERT mehrerer Reihen:

```
...  
INSERT INTO SQL-RETIREE (NAME,AGE,SEX)  
SELECT LASTNAME, AGE, SEX  
FROM SQL-EMPLOYEES  
WHERE AGE > 60  
...
```

Anmerkung:

Die Anzahl der Reihen, die tatsächlich eingefügt wurden, kann mit der Systemvariablen *ROWCOUNT (siehe Natural Referenzhandbuch) ermittelt werden.

PROCESS SQL

```
PROCESS SQL ddm-name << statement-string >>
```

Funktion

Das Statement PROCESS SQL dient dazu, mit SQL-Statements auf eine Datenbank zuzugreifen.

ddm-name

Mit dem DDM-Namen geben Sie an, für welche Datenbank die angegebenen SQL-Statements ausgeführt werden sollen. Weitere Informationen zu *ddm-name* finden Sie auf Seite 680.

statement-string

Die Statements, die Sie im *statement-string* angeben können, sind dieselben, die Sie auch mit dem SQL-Statement EXECUTE (vgl. **Flexible SQL**, Seite 712) ausführen können.

Anmerkung:

Um Transaktionssynchronisationsprobleme zwischen Natural und der Datenbank zu vermeiden, dürfen die Statements COMMIT und ROLLBACK nicht im PROCESS SQL-Statement verwendet werden.

Der *statement-string* kann über mehrere Zeilen gehen, ohne daß am Zeilenende ein Fortsetzungszeichen erforderlich ist. Er kann ganze oder teilweise Kommentarzeilen enthalten.

Der *statement-string* darf auch Parameter enthalten.

Parameter

$\begin{bmatrix} :U \\ :G \end{bmatrix} :host-variable \text{ [INDICATOR :host-variable] [LINDICATOR :host-variable]}$
--

Im Gegensatz zu den auf Seite 684 beschriebenen *parameters* muß hier den *host-variables* ein Doppelpunkt (:) vorangestellt werden. Außerdem kann ihnen ein weiterer Qualifizierer (“:U” bzw. “:G”) vorangestellt werden.

Näheres zu *host-variable* finden Sie auf Seite 684.

:U:*host-variable*

Der Präfix “:U” qualifiziert die *host-variable* als sogenannte “Using”-Variable; d.h. ihr Wert wird an die Datenbank *übergeben*. “:U” ist der Standardpräfix.

:G:*host-variable*

Der Präfix “:G” qualifiziert die *host-variable* als sogenannte “Giving”-Variable; d.h. sie *erhält* einen Wert *von der* Datenbank.

Beispiele

Beispiele für DB2 (unter OS/390):

```
PROCESS SQL DB2_DDM << CONNECT TO :LOCATION >>
```

```
PROCESS SQL DB2_DDM << SET :G:LOCATION = CURRENT SERVER >>
```

Beispiel für Adabas D:

```
PROCESS SQL ADABAS_D_DDM << LOCK TABLE EMPLOYEES IN SHARE MODE >>
```

Beispiel für Aufruf einer in Adabas D gespeicherten Prozedur:

Die aufgerufene Prozedur berechnet die Summe zweier Zahlen.

```
...  
COMPUTE #N1 = 1  
COMPUTE #N2 = 2  
COMPUTE #SUM = 0  
...  
PROCESS SQL ADABAS_D_DDM << DBPROCEDURE DEMO.SUM (:#N1, :#N2, :G:#SUM) >>  
...  
WRITE #N1 '+' #N2 ' =' #SUM  
...
```

Optionen für Entire Access

Mit Entire Access können Sie als *statement-string* auch folgendes angeben:

- **SET SQLOPTION** *option = value*
- **SQLCONNECT** *option = value*
- **SQLDISCONNECT**

Diese Optionen sind nur mit Entire Access möglich und in der Dokumentation für Entire Access beschrieben.

READ RESULT SET

```
READ [(limit)] RESULT SET result-set INTO {VIEW view-name} FROM ddm-name
      {parameter,...}
      [GIVING sqlcode]
END-RESULT
```

Funktion

Das Statement READ RESULT SET dient dazu, einen Result Set zu lesen, der von einer mit einem vorhergehenden CALLDBPROC-Statement aufgerufenen Stored Procedure erzeugt wurde.

Das READ RESULT SET-Statement kann nur in Verbindung mit einem CALLDBPROC-Statement verwendet werden.

Als *result-set* geben Sie eine Result-Set-Locator-Variable an, die von einem vorhergehenden CALLDBPROC-Statement gefüllt wurde. *Result-set* muß eine Variable von Format/Länge I4 sein.

Anmerkung:

Falls zwischen dem CALLDBPROC-Statement und dem READ RESULT SET-Statement eine Syncpoint-Operation stattfand, kann das READ RESULT SET-Statement nicht mehr auf die Result Sets zugreifen.

limit

Sie können die Anzahl der zu lesenden Reihen begrenzen. Sie können das *limit* entweder als numerische Konstante (0 bis 99999999) oder als Variable von Format N, P oder I angeben.

ddm-name

Als *ddm-name* geben Sie den Namen des DDMs an, das benutzt wird, um die Datenbank zu "adressieren", die die Stored Procedure ausführt. Weitere Informationen zu *ddm-name* siehe Seite 680.

GIVING *sqlcode*

Mit dieser Option erhalten Sie den SQL-Code der SQL-“Fetch“-Operation, mit der der Result Set verarbeitet wurde.

Wenn Sie diese Option angeben und der SQL-Code der SQL-Operation ist nicht “0”, wird keine Natural-Fehlermeldung ausgegeben. In diesem Fall muß die als Reaktion auf den SQL-Codewert auszuführende Handlung im aufrufenden Natural-Objekt programmiert werden.

Das *sqlcode*-Feld muß eine Variable von Format/Länge I4 sein.

Wenn Sie die Option **GIVING** *sqlcode* nicht verwenden, gibt Natural eine Fehlermeldung aus, falls der SQL-Code der Stored Procedure nicht “0” ist.

Beispiel

Siehe entsprechendes Beispiel beim CALLDBPROC-Statement.

Siehe entsprechendes Natural-Datenbankschnittstellen-Handbuch in der Dokumentation für Natural für Großrechner.

ROLLBACK

ROLLBACK

Funktion

Das SQL-Statement ROLLBACK entspricht dem Natural-Statement BACKOUT TRANSACTION. Es macht alle seit dem Beginn der letzten "recovery unit" ausgeführten Datenbankänderungen rückgängig. Eine "recovery unit" beginnt entweder zu Beginn der Session oder nach einem SYNCPOINT-, COMMIT-, END TRANSACTION- oder BACKOUT TRANSACTION-Statement. Außerdem bewirkt ROLLBACK, daß alle während der Transaktion gehaltenen Datensätze freigegeben werden.

Bei dem Versuch, Datenbankänderungen, die bereits durch einen Terminal-I/O bestätigt wurden, mit ROLLBACK wieder rückgängig zu machen, gibt Natural Fehlermeldung NAT3711 aus.

Beispiel:

```
...  
DELETE FROM SQL-PERSONNEL WHERE NAME = 'SMITH'  
ROLLBACK  
...
```

Da bei Beendigung einer logischen Arbeitseinheit alle Cursor geschlossen werden, darf ein ROLLBACK-Statement nicht innerhalb einer datenbankverändernden Verarbeitungsschleife stehen, sondern muß nach einer solchen stehen (bzw. bei geschachtelten Schleifen nach der äußersten Schleife).

Hinweis für Nicht-Natural-Programme

Wenn ein Natural-Programm ein Nicht-Natural-Programm aufruft, sollte das aufgerufene Programm kein eigenes ROLLBACK-Statement enthalten, falls das aufrufende Natural-Programm selbst auch Datenbankaufrufe durchführt. In diesem Falle sollte das Natural-Programm das ROLLBACK-Statement für das Nicht-Natural-Programm enthalten.

SELECT

Gemäß der Standard-SQL-Funktionalität unterstützt Natural sowohl das cursor-orientierte SELECT, mit dem eine beliebige Anzahl von Reihen gelesen werden kann, als auch das nicht cursor-orientierte “singleton” SELECT, das maximal eine Reihe liest.

Mit der Konstruktion “SELECT ... END-SELECT” verwendet Natural die gleiche Datenbankschleifen-Verarbeitung wie beim FIND-Statement.

Cursor-orientierte Auswahl

```

SELECT selection INTO { parameter, ...
                       VIEW {view-name [correlation-name]},... } table-expression

[ { UNION
  EXCEPT
  INTERSECT } [ALL][()] SELECT selection table-expression [] ] ...

[ ORDER BY { integer
             column-reference } [ ASC
                                 DESC ] ]

[ OPTIMIZE FOR integer ROWS ]

[ WITH { CS
        RR
        UR } ]

[ WITH HOLD ]

[ WITH RETURN ]

[ IF-NO-RECORDS-FOUND-clause ]
statement...

{ END-SELECT
  LOOP (nur Reporting Mode) }

```

Das cursor-orientierte SELECT-Statement dient (wie das FIND-Statement) dazu, ausgehend von einem Suchkriterium Reihen (rows) von einer oder mehreren Datenbanktabellen auszuwählen. Darüber hinaus wird die Cursor-Verwaltung von Natural automatisch erledigt und muß daher nicht mehr im Anwendungsprogramm kodiert werden.

Nicht cursor-orientierte Auswahl

SELECT SINGLE

selection INTO { *parameter,...*
VIEW {*view-name* [*correlation-name*]},... } *table-expression*

[**WITH** { **CS**
RR
UR }]

[*IF-NO-RECORDS-FOUND-clause*]

statement...

{ **END-SELECT**
LOOP (*nur Reporting Mode*) }

SELECT SINGLE unterstützt die Funktionalität eines “singleton SELECT”, das keine Cursor verwendet und maximal eine Reihe liest. Es kann nicht von einem “positioned” UPDATE- bzw. DELETE-Statement referenziert werden.

table-expression

Die *table-expression* besteht aus einer FROM-Klausel und einer optionalen WHERE-Klausel. Die GROUP BY- und HAVING-Klauseln sind nicht erlaubt.

Example 1:

```
DEFINE DATA LOCAL
01 #NAME      (A20)
01 #FIRSTNAME (A15)
01 #AGE       (I2)
...
END-DEFINE
...
SELECT NAME, FIRSTNAME, AGE
      INTO #NAME, #FIRSTNAME, #AGE
      FROM SQL-PERSONNEL
      WHERE NAME IS NOT NULL
            AND AGE > 20
...
      DISPLAY #NAME #FIRSTNAME #AGE
END-SELECT
...
END
```

Example 2:

```
DEFINE DATA LOCAL
01 #COUNT   (I4)
...
END-DEFINE
...
SELECT SINGLE COUNT(*) INTO #COUNT FROM SQL-PERSONNEL
...
```

Näheres zu *selection* und *table-expression* siehe Seite 705 bzw. Seite 707.

Anmerkung:

Im folgenden wird der Begriff “SELECT-Statement” verwendet im Sinne einer vollständigen “query-expression”, die aus mehreren mit UNION verknüpften “SELECT-expressions” besteht.

INTO-Klausel

$$\text{INTO} \left\{ \text{VIEW } \left\{ \begin{array}{l} \textit{parameter, ...} \\ \textit{view-name} [\textit{correlation-name}] \end{array} \right\}, \dots \right\}$$

In der INTO-Klausel geben Sie die Zielfelder im Programm an, die mit dem Ergebnis der Abfrage gefüllt werden sollen. Sie können in der INTO-Klausel entweder einzelne *parameter* oder ganze im DEFINE DATA-Statement definierte Views angeben.

Die Zielfelder können aus einer einzigen Tabelle kommen bzw. bei einem "Join" (vgl. Abschnitt "**Join**"-Abfragen, Seite 748) auch aus mehreren.

Anmerkung:

In der Standard-SQL-Syntax wird die INTO-Klausel nur in nicht cursor-orientierten "singleton SELECT"-Operationen verwendet, bei denen eine einzelne Reihe gelesen werden soll. Natural erlaubt die INTO-Klausel jedoch sowohl in cursor-orientierten als auch in nicht cursor-orientierten SELECT-Operationen.

Die *selection* kann auch aus nur einem Stern (*) bestehen. In einer standardmäßigen *selection-expression* steht dieser für eine Liste aller Spaltennamen der in der FROM-Klausel angegebenen Tabelle(n). Im Natural-SELECT-Statement hat der Ausdruck "SELECT *" jedoch eine andere Bedeutung: Alle in der INTO-Klausel gemachten Angaben werden auch in der *selection* verwendet, ohne daß sie dort explizit angegeben werden müßten. Ihre Namen müssen vorhandenen Datenbank-Spaltennamen entsprechen.

Beispiele:

```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
  02 NAME
  02 AGE
END-DEFINE
```

```
...
SELECT *
  INTO NAME, AGE
```

```
...
SELECT *
  INTO VIEW PERS
```

Obige Beispiele sind mit den folgenden gleichwertig:

```
...
SELECT NAME, AGE
  INTO NAME, AGE
```

```
...
SELECT NAME, AGE
  INTO VIEW PERS
```

parameter

Wenn Sie einzelne *parameter* als Zielfelder angeben, müssen sie in Anzahl und Format mit den in der entsprechenden *selection* angegebenen *columns* bzw. *scalar-expressions* übereinstimmen wie oben beschrieben (Näheres zu *scalar-expression* siehe Seite 692).

Beispiel:

```
DEFINE DATA LOCAL
01 #NAME    (A20)
01 #AGE     (I2)
END-DEFINE
...
SELECT NAME, AGE
      INTO #NAME, #AGE
      FROM SQL-PERSONNEL
...
```

Die Zielfelder #NAME und #AGE, die Natural-Programmvariablen sind, erhalten den Inhalt der Datenbankspalten NAME und AGE.

VIEW-Klausel

```
VIEW {view-name [correlation-name]},...
```

Wenn in der INTO-Klausel ein oder mehrere Views referenziert werden, muß die Anzahl der in der *selection* gemachten Angaben der Anzahl der in dem/den View(s) definierten Felder entsprechen (hierbei werden Gruppenfelder, redefinierte Felder und Indikatorfelder nicht mitgezählt).

Anmerkung:

Die Natural-Zielfelder wie auch die Tabellenspalten müssen im Natural-DDM definiert sein (wobei die Namen allerdings unterschiedlich sein dürfen, da die Zuweisung entsprechend ihrer Reihenfolge erfolgt).

Beispiel einer INTO-Klausel mit View:

```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
   02 NAME
   02 AGE
END-DEFINE
...
SELECT FIRSTNAME, AGE
   INTO VIEW PERS
   FROM SQL-PERSONNEL
...
```

Die Zielfelder NAME und AGE, die Teil eines Natural-Views sind, erhalten den Inhalt der Datenbankspalten FIRSTNAME und AGE.

correlation-name

Wenn die VIEW-Klausel in einem “SELECT *” verwendet wird, in dem mehrere Tabellen mit “JOIN” verknüpft werden, sind *correlation-names* erforderlich, falls der angegebene View Felder enthält, die Spalten referenzieren, welche in mehreren dieser Tabellen vorkommen. Um zu bestimmen, von welcher Spalte ausgewählt werden soll, werden bei der Generierung der Auswahlliste alle diese Spalten mit dem angegebenen *correlation-name* qualifiziert. Der einem View zugewiesene *correlation-name* muß einem der *correlation-names* entsprechen, mit denen die verknüpften Tabellen qualifiziert werden. Vgl. Abschnitt “**Join**”-Abfragen (Seite 748).

Beispiel:

```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
    02 NAME
    02 FIRST-NAME
    02 AGE
END-DEFINE
...
SELECT *
    INTO VIEW PERS A
    FROM SQL-PERSONNEL A, SQL-PERSONNEL B
...
```

Abfrage mit UNION

UNION vereinigt die Ergebnisse von zwei oder mehr *select-expressions* miteinander. Die in den einzelnen *select-expressions* angegebenen Spalten müssen UNION-kompatibel sein, d.h. in Anzahl, Typ und Format zueinander passen.

Redundante doppelte Reihen werden immer aus dem Ergebnis einer UNION eliminiert, es sei denn, der UNION-Operator enthält ausdrücklich ein ALL. Allerdings ist es bei UNION nicht möglich, DISTINCT explizit als Alternative zu ALL anzugeben.

Beispiel:

```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
    02 NAME
    02 AGE
    02 ADDRESS (1:6)
END-DEFINE
...
SELECT NAME, AGE, ADDRESS
    INTO VIEW PERS
    FROM SQL-PERSONNEL
    WHERE AGE > 55
UNION ALL
SELECT NAME, AGE, ADDRESS
    FROM SQL-EMPLOYEES
    WHERE PERSNR < 100
ORDER BY NAME
...
END-SELECT
...
```

Grundsätzlich ist die Anzahl der *select-expressions*, die mit UNION verknüpft werden können, beliebig.

Nur die erste *select-expression* darf eine INTO-Klausel enthalten.

Wird eine ORDER BY-Klausel verwendet, muß sie nach der letzten *selection-expression* angegeben werden. Die zu sortierenden Spalten müssen durch die Spaltennummern identifiziert werden, nicht durch die Spaltennamen.

ORDER BY-Klausel

$$\text{ORDER BY} \left\{ \begin{array}{l} \textit{integer} \\ \textit{column-reference} \end{array} \right\} \left[\begin{array}{l} \text{ASC} \\ \text{DESC} \end{array} \right] , \dots$$

Die ORDER BY-Klausel sortiert das Ergebnis der Abfrage in einer bestimmten Reihenfolge.

In jeder ORDER BY-Klausel muß eine Spalte der Ergebnistabelle angegeben werden. In den meisten ORDER BY-Klauseln wird die Spalte entweder durch eine *column-reference* (also den optional qualifizierten Spaltennamen) oder durch die Spaltennummer identifiziert. In einer Abfrage mit UNION muß eine Spalte durch die Spaltennummer identifiziert werden. Die Spaltennummer ist die Ordinalzahl, die die Position einer Spalte (von links nach rechts) innerhalb der *selection* angibt, also eine Ganzzahl (*integer*). Dadurch ist es möglich, ein Ergebnis auf der Grundlage einer berechneten Spalte, die keinen Namen hat, zu sortieren.

Beispiel:

```
DEFINE DATA LOCAL
1 #NAME          (A20)
1 #YEARS-TO-WORK (I2)
END-DEFINE
...
SELECT NAME , 65 - AGE
      INTO #NAME, #YEARS-TO-WORK
      FROM SQL-PERSONNEL
      ORDER BY 2
...
```

Als Sortierreihenfolge können Sie entweder aufsteigend (ASC = ascending) oder absteigend (DESC = descending) angeben. Standardmäßig gilt ASC.

Beispiel:

```
DEFINE DATA LOCAL
1 PERS VIEW OF SQL-PERSONNEL
1 NAME
1 AGE
1 ADDRESS (1:6)
END-DEFINE
...
SELECT NAME, AGE, ADDRESS
  INTO VIEW PERS
  FROM SQL-PERSONNEL
  WHERE AGE = 55
  ORDER BY NAME DESC
...
```

Näheres zu *integer* und *column-reference* siehe Seite 679 bzw. Seite 693.

IF NO RECORDS FOUND-clause

Diese Klausel ist eigentlich nicht Bestandteil von Natural SQL; sie stellt eine Natural-Funktion dar, die für SQL-Schleifenverarbeitung zur Verfügung gestellt wird.

Structured-Mode-Syntax

```
IF NO [RECORDS] [FOUND]
```

```
{ ENTER  
  statement ... }
```

```
END—NOREC
```

Reporting-Mode-Syntax

```
IF NO [RECORDS] [FOUND]
```

```
{ ENTER  
  statement  
  DO statement ... DOEND }
```

In der IF NO RECORDS FOUND-Klausel können Sie eine Verarbeitung angeben, die ausgeführt werden soll für den Fall, daß kein Datensatz die im vorangegangenen SELECT-Statement angegebenen Selektionskriterien erfüllt.

Wenn kein Datensatz die angegebenen Selektionskriterien erfüllt, dann löst die IF NO RECORDS FOUND-Klausel eine Verarbeitungsschleife aus, die einmal mit einem "leeren" Datensatz durchlaufen wird. Falls Sie dies nicht wünschen, geben Sie in der IF NO RECORDS FOUND-Klausel das Statement ESCAPE BOTTOM an.

Enthält die IF NO RECORDS FOUND-Klausel ein oder mehrere Statements, werden diese ausgeführt, unmittelbar bevor die Schleife durchlaufen wird. Sollen vor Durchlaufen der Schleife keine weiteren Statements ausgeführt werden, muß die IF NO RECORDS FOUND-Klausel das Schlüsselwort ENTER enthalten.

Anmerkung:

*Falls das Ergebnis-Set des SELECT-Statements aus einer einzelnen Reihe von NULL-Werten besteht, wird die IF NO RECORDS FOUND-Klausel **nicht** ausgeführt. Dies kann der Fall sein, wenn die "selection"-Liste nur aus einer der "aggregate-functions" SUM, AVG, MIN oder MAX auf Spalten besteht und der Set, mit dem diese "aggregate-functions" operieren, leer ist.*

Bei obengenannter Verwendung dieser "aggregate-functions" sollten Sie daher keine IF NO RECORDS FOUND-Klausel benutzen, sondern stattdessen die Werte der betreffenden Null-Indikator-Felder abfragen.

Datenbankwerte

Natural setzt alle Datenbankfelder, die die in der aktuellen Verarbeitungsschleife angegebene Datei referenzieren, auf Leerwerte, es sei denn, eines der in der IF NO RECORDS FOUND-Klausel angegebenen Statements weist den Feldern andere Werte zu.

Auswertung von Systemfunktionen

Natural-Systemfunktionen werden einmal für den leeren Datensatz ausgewertet, der für die aus der IF NO RECORDS FOUND-Klausel resultierende Verarbeitung erstellt wurde.

“Join”-Abfragen

Ein “Join” ist eine Abfrage, bei der Daten von mehr als einer Tabelle gelesen werden. Alle betroffenen Tabellen müssen in der FROM-Klausel angegeben werden.

Beispiel:

```
DEFINE DATA LOCAL
1 #NAME      (A20)
1 #MONEY     (I4)
END-DEFINE
...
SELECT NAME, ACCOUNT
      INTO #NAME, #MONEY
      FROM SQL-PERSONNEL P, SQL-FINANCE F
      WHERE P.PERSNR = F.PERSNR
            AND F.ACCOUNT > 10000
...

```

Ein “Join” bildet immer zunächst das kartesische Produkt der in der FROM-Klausel angegebenen Tabellen und eliminiert später von diesem kartesischen Produkt alle Reihen, die die in der WHERE-Klausel angegebene “Join”-Bedingung nicht erfüllen.

Bei längeren Tabellennamen können Sie sich durch Verwendung von *correlation-names* Schreibarbeit sparen. Wird in einer “Join”-Abfrage eine Tabelle mit sich selbst verknüpft, ist die Angabe von *correlation-names* erforderlich, um die beiden nötigen Referenzen auf dieselbe Tabelle voneinander zu unterscheiden (vgl. Seite 683).

UPDATE

Syntax 1 — Searched UPDATE

$\text{UPDATE} \left\{ \begin{array}{l} \text{view-name [correlation-name] SET *} \\ \text{table-name [correlation-name] SET assignment-list} \end{array} \right\}$ <p style="text-align: center;">[WHERE search-condition]</p>

Syntax 2 — Positioned UPDATE

$\text{UPDATE} \left\{ \begin{array}{l} \text{view-name SET *} \\ \text{table-name SET assignment-list} \end{array} \right\} \text{WHERE CURRENT OF CURSOR [(r)]}$
--

Funktion

Das SQL-Statement UPDATE dient dazu, Reihen in einer Tabelle zu ändern, ohne einen Cursor zu verwenden (“searched” UPDATE), oder Spalten in der Reihe zu ändern, auf die der Cursor zeigt (“positioned” UPDATE).

“Searched UPDATE” ist ein eigenständiges Statement, das unabhängig von einem SELECT-Statement verwendet werden kann. Mit einem einzigen Statement können Sie keine, eine, mehrere oder alle Reihen einer Tabelle ändern. Welche Reihen geändert werden, bestimmen Sie mit der Suchbedingung (*search-condition*), die auf die Tabelle angewandt wird. Außerdem ist es möglich, einem Tabellen- oder View-Namen einen *correlation-name* zuzuweisen.

Ein “positioned UPDATE” bezieht sich auf einen Cursor innerhalb einer Datenbankschleife. Es muß daher dieselbe Tabelle bzw. denselben View referenzieren wie das entsprechende SELECT-Statement, sonst erfolgt eine Fehlermeldung. Ein “positioned UPDATE” kann nur mit cursor-orientierter Selektion verwendet werden.

Näheres zu *view-name*, *table-name*, *authorization-identifier* und *correlation-name* finden Sie unter **Grundlegende Syntaxbestandteile** (Seite 679).

Anmerkung:

*Die Anzahl der Reihen, die mit einem “searched UPDATE” tatsächlich geändert wurden, kann mit der Systemvariablen *ROWCOUNT (siehe Natural Referenzhandbuch) ermittelt werden.*

SET-Klausel

Wenn sich die Änderungen auf einen View beziehen, müssen Sie in der SET-Klausel einen Stern (*) angeben, da alle Spalten des Views geändert werden müssen.

Wenn sich die Änderungen auf eine Tabelle beziehen, können Sie in der SET-Klausel entweder eine *assignment-list* angeben oder den Namen eines Views, der die zu ändernden Spalten enthält.

assignment-list

$$\left\{ \text{column-name} = \left\{ \begin{array}{c} \text{scalar-expression} \\ \text{NULL} \end{array} \right\} \right\}, \dots$$

In einer *assignment-list* können Sie einer oder mehreren Spalten Werte zuweisen. Ein Wert kann entweder eine *scalar-expression* oder NULL sein.

Näheres zu *scalar-expressions* siehe Seite 692.

Wenn Sie NULL zuweisen, bedeutet dies, daß das betreffende Feld keinen Wert enthalten soll (auch nicht den Wert "0" oder "Leerzeichen").

WHERE *search-condition*

In der WHERE-Klausel geben Sie eine Suchbedingung (*search-condition*) an, die bestimmt, welche Reihen geändert werden sollen.

Wenn Sie keine WHERE-Klausel angeben, wird die gesamte Tabelle geändert.

Statement-Referenzierung (*r*)

Die Notation "(*r*)" dient zur Referenzierung des Statements, das zur Selektion der zu ändernden Reihe verwendet wurde. Wird keine Statement-Referenz angegeben, bezieht sich das UPDATE-Statement auf die jeweils innerste aktive Verarbeitungsschleife, mit der die Reihe, die geändert werden soll, ausgewählt wurde.

Beispiele

Beispiel für Searched UPDATE:

```
DEFINE DATA LOCAL
1 PERS VIEW OF SQL-PERSONNEL
  2 NAME
  2 AGE
...
END-DEFINE
...
ASSIGN AGE = 45
ASSIGN NAME = 'SCHMIDT'
UPDATE PERS SET * WHERE NAME = 'SCHMIDT'
...
```

Beispiel für Searched UPDATE mit *assignment-list*:

```
DEFINE DATA LOCAL
1 PERS VIEW OF SQL-PERSONNEL
  2 NAME
  2 AGE
...
END-DEFINE
...
UPDATE SQL-PERSONNEL SET AGE = AGE + 1 WHERE NAME = 'SCHMIDT'
...
```

Beispiel für Positioned UPDATE:

```
DEFINE DATA LOCAL
1 PERS VIEW OF SQL-PERSONNEL
  2 NAME
  2 AGE
...
END-DEFINE
...
SELECT * INTO PERS FROM SQL_PERSONNEL WHERE NAME = 'SCHMIDT'
  COMPUTE AGE = AGE + 1
  UPDATE PERS SET * WHERE CURRENT OF CURSOR
END-SELECT
...
```

Beispiel für Positioned UPDATE mit *assignment-list*:

```
DEFINE DATA LOCAL
1 PERS VIEW OF SQL-PERSONNEL
  2 NAME
  2 AGE
...
END-DEFINE
...
SELECT * INTO PERS FROM SQL-PERSONNEL WHERE NAME = 'SCHMIDT'
  UPDATE SQL-PERSONNEL SET AGE = AGE + 1 WHERE CURRENT OF CURSOR
END-SELECT
...
```

INDEX

A

Abbrechen

- Natural-Session, TERMINATE-Statement, 641
- Programm, STOP-Statement, 626
- Transaktion, BACKOUT TRANSACTION-Statement, 51
- Verarbeitungsschleife, ESCAPE-Statement, 278

Abend. *Siehe* Abbrechen

ACCEPT/REJECT-Statements, 16

ADABAS, Interne Satznummer. *Siehe* ISN

Addition

- Siehe auch* Arithmetik
- ADD-Statement, 20
- COMPUTE-Statement, 137

ADD-Statement, 20

AIV, 157, 177

Alte Statements, SQL, 675

Anfang, einer Ausgabeseite, AT TOP OF PAGE-Statement, 47

Anwendung. *Siehe* Library

Anzeigen, von Daten

- DISPLAY-Statement, 243
- PRINT-Statement, 468
- WRITE-Statement, 649

Applikation. *Siehe* Library

Applikationsunabhängige Variable. *Siehe* AIV

Arbeitsdatei

- beschreiben, WRITE WORK FILE-Statement, 670
- Dateinamen zuweisen, DEFINE WORK FILE-Statement, 225
- lesen, READ WORK FILE-Statement, 526
- schließen, CLOSE WORK FILE-Statement, 103
- Statements, Übersicht, 13

Arithmetik

- ADD-Statement, 20
- COMPUTE-Statement, 137
- DIVIDE-Statement, 261
- MULTIPLY-Statement, 435
- SUBTRACT-Statement, 635

Natural Statements Handbuch

Array

- Ausgangswert für, DEFINE DATA-Statement, 175
- Ausprägungen, lesen, OBTAIN-Statement, 444
- definieren, DEFINE DATA-Statement, 173

ASSIGN-Statement. *Siehe* COMPUTE-Statement

AT BREAK-Statement, 24

AT END OF DATA-Statement, 33

AT END OF PAGE-Statement, 37

AT START OF DATA-Statement, 43

AT TOP OF PAGE-Statement, 47

Aufrufen

Nicht-Natural-Programm

CALL FILE-Statement, 84

CALL LOOP-Statement, 87

CALL-Statement, 58

Programm, FETCH-Statement, 294

Subprogramm, CALLNAT-Statement, 89

Subroutine, PERFORM-Statement, 457

Aufteilen von Daten, SEPARATE-Statement, 585

Ausdrucken. *Siehe* Drucken

Ausführen

Nicht-NATURAL-Programm, CALL FILE-Statement, 84

Nicht-Natural-Programm

CALL LOOP-Statement, 87

CALL-Statement, 58

Programm, RUN-Statement, 573

Subprogramm, CALLNAT-Statement, 89

Subroutine, PERFORM-Statement, 457

Ausgabe

Siehe auch Map

von Daten

DISPLAY-Statement, 243

PRINT-Statement, 468

WRITE-Statement, 649

Zeilenvorschub, SKIP-Statement, 611

Ausgangswert, für Array, DEFINE DATA-Statement, 175

Ausgangswert einer Variablen

DEFINE DATA-Statement, 156

Füllzeichen, DEFINE DATA-Statement, 171, 176

RESET-Statement, 568

Ausprägung. *Siehe* Array

B

Backout, aus einer Transaktion, BACKOUT TRANSACTION-Statement, 51
BACKOUT TRANSACTION-Statement, 51
Bedingte Verarbeitung
 DECIDE FOR-Statement, 145
 DECIDE ON-Statement, 149
 IF SELECTION-Statement, 366
 IF-Statement, 362
Beenden. *Siehe* Abbrechen; Ende
BEFORE BREAK PROCESSING-Statement, 55
Benutzervariablen, definieren, DEFINE DATA-Statement, 156
Berechnen. *Siehe* Arithmetik
Bildschirmfenster. *Siehe* Window
Bildschirmmaske. *Siehe* Map
Block. *Siehe* DEFINE DATA-Statement
Break. *Siehe* Gruppenwechsel
BY VALUE Parameter-Definition, 161

C

CALL FILE-Statement, 84
CALL LOOP-Statement, 87
CALLDBPROC-Statement, 716
CALLNAT-Statement, 89
CALL-Statement, 58
 INTERFACE4, 70
CLOSE CONVERSATION-Statement, 98
CLOSE DIALOG-Statement, 100
CLOSE PC-Statement, 101
CLOSE PRINTER-Statement, 101
CLOSE WORK FILE-Statement, 103
COMMIT-Statement, 721
Common Set, SQL, 678
COMPOSE-Statement, 105
COMPRESS-Statement, 130
COMPUTE-Statement, 137
CON-FORM, Textformatierung, COMPOSE-Statement, 105
CON-NECT, Textformatierung, COMPOSE-Statement, 105
Copycode, INCLUDE-Statement, 370
Coupling von Dateien, FIND-Statement, 317
CREATE OBJECT-Statement, 143

D

Data Area, definieren, DEFINE DATA-Statement, 156

Data Definition Module. *Siehe* DDM

Daten

-änderung

UPDATE-Statement, 643

UPDATE-Statement (SQL), 749

anzeigen

DISPLAY-Statement, 243

PRINT-Statement, 468

WRITE-Statement, 649

aufteilen, SEPARATE-Statement, 585

-bank. *Siehe* Datensatz; Transaktion

-bereich. *Siehe* Data Area

-block, DEFINE DATA-Statement, 159

-definitionsmodul. *Siehe* DDM

im Stack ablegen, STACK-Statement, 621

-satz. *Siehe* Datensatz

übertragen in anderes Feld

COMPUTE-Statement, 137

MOVE ALL-Statement, 432

MOVE-Statement, 419

SEPARATE-Statement, 585

- Datensatz
 - ändern
 - UPDATE-Statement, 643
 - UPDATE-Statement (SQL), 749
 - akzeptieren/zurückweisen, ACCEPT/REJECT-Statements, 16
 - im Hold, RETRY-Statement, 571
 - lesen
 - FIND-Statement, 299
 - GET SAME-Statement, 348
 - GET-Statement, 344
 - HISTOGRAM-Statement, 355
 - READ-Statement, 511
 - SELECT-Statement, 735
 - löschen
 - DELETE-Statement, 239
 - DELETE-Statement (SQL), 722
 - sortieren, SORT-Statement, 613
 - speichern
 - INSERT-Statement, 724
 - STORE-Statement, 628
- DDM, DEFINE DATA-Statement, 166
- DECIDE FOR-Statement, 145
- DECIDE ON-Statement, 149
- DEFINE CLASS-Statement, 153
- DEFINE DATA-Statement, 156
 - HANDLE-Definition, 164
 - Parameter BY VALUE Definition, 161
 - PARAMETER-HANDLE-Definition, 164
- DEFINE PRINTER-Statement, 188
- DEFINE SUBROUTINE-Statement, 208
- DEFINE WINDOW-Statement, 214
- DEFINE WORK FILE-Statement, 225
- DELETE-Statement, 239
- DELETE-Statement (SQL), 722
- Device. *Siehe* Terminal
- DISPLAY-Statement, 243
- DIVIDE-Statement, 261
- Division
 - Siehe auch* Arithmetik
 - COMPUTE-Statement, 137
 - DIVIDE-Statement, 261
- DO/DOEND-Statements, 265

Natural Statements Handbuch

DOWNLOAD-Statement, 267
Drucken, PRINT-Statement, 468
Drucker
 CLOSE PRINTER-Statement, 101
 DEFINE PRINTER-Statement, 188
Dynamische Sourcecode-Generierung, RUN-Statement, 574

E

Editiermaske, DEFINE DATA-Statement, 177
Eingabe. *Siehe* Input
EJECT-Statement, 268
END TRANSACTION-Statement, 273
Ende
 der gelesenen Daten, AT END OF DATA-Statement, 33
 einer Ausgabeseite, AT END OF PAGE-Statement, 37
 einer Transaktion, END TRANSACTION-Statement, 273
 einer Verarbeitungsschleife, LOOP-Statement, 412
 eines Programms, END-Statement, 272
End-of-Data-Verarbeitung, AT END OF DATA-Statement, 33
End-of-Page-Verarbeitung, AT END OF PAGE-Statement, 37
END-Statement, 272
ENTER-Taste. *Siehe* Funktionstasten
Entire System Server, PROCESS-Statement, 477
Erweitern, EXPAND-Statement, 293
ESCAPE-Statement, 278
EXAMINE TRANSLATE-Statement, 290
EXAMINE-Statement, 282
EXPAND-Statement, 293
Extended Set, SQL, 678
Externe Subroutine. *Siehe* Subroutine

F

Fehler, -verarbeitung
 ON ERROR-Statement, 445
 REINPUT-Statement, 537

Feld

- definieren, DEFINE DATA-Statement, 156
- Inhalt übertragen, COMPRESS-Statement, 130
- redefinieren
 - DEFINE DATA-Statement, 167
 - REDEFINE-Statement, 533
- wert
 - Ausgabe identischer unterdrücken, SUSPEND IDENTICAL SUPPRESS-Statement, 638
 - untersuchen, EXAMINE-Statement, 282
 - zurücksetzen, RESET-Statement, 568
- Fenster. *Siehe* Window
- FETCH-Statement, 294
- FILLER-Option, DEFINE DATA-Statement, 167
- FIND-Statement, 299
- Flexible SQL, 712
- Format, von Operanden, 6
- Formatieren von Text, COMPOSE-Statement, 105
- FORMAT-Statement, 340
- FOR-Statement, 337
- Funktionstasten, Funktion zuweisen, SET KEY-Statement, 596
- Fußzeile, WRITE TRAILER-Statement, 665

G

- GET SAME-Statement, 348
- GET TRANSACTION DATA-Statement, 352
- GET-Statement, 344
- Global Data Area. *Siehe* Data Area
- Globale Variablen, freigeben, RELEASE-Statement, 548
- Globaler Datenbereich. *Siehe* Data Area
- Großbuchstaben, Umsetzung in, EXAMINE TRANSLATE-Statement, 290
- Großrechner, Begriffsdefinition, 2
- Gruppe, in Datendefinition, DEFINE DATA-Statement, 159
- Gruppenwechsel, Verarbeitung
 - AT BREAK-Statement, 24
 - BEFORE BREAK PROCESSING-Statement, 55
 - PERFORM BREAK PROCESSING-Statement, 466

H

- HANDLE-Definition, 164

Natural Statements Handbuch

Helproutine, aufrufen, REINPUT-Statement, 537
HISTOGRAM-Statement, 355
Hold, Datensatz im, RETRY-Statement, 571

I

Identical Suppress, SUSPEND IDENTICAL SUPPRESS-Statement, 638
Identische Feldwerte unterdrücken. *Siehe* Identical Suppress
IF SELECTION-Statement, 366
IF-Statement, 362
IGNORE-Statement, 369
INCLUDE-Statement, 370
Index. *Siehe* Array
Infoline, DEFINE PRINTER-Statement, 204
Initialwert einer Variablen
 DEFINE DATA-Statement, 169
 RESET-Statement, 568
Inline-Subroutine. *Siehe* Subroutine
INPUT-Statement, 374
INSERT-Statement, 724
INTERFACE4, CALL-Statement, 70
Interfaces, INTERFACE-Statement, 399
INTERFACE-Statement, 399
Interne Satznummer. *Siehe* ISN
Interne Subroutine. *Siehe* Subroutine
ISN
 freigeben, RELEASE-Statement, 548
 Lesen nach, READ-Statement, 516
 lesen nach, GET-Statement, 344
 zurückhalten, FIND-Statement, 323

J

Join-Abfrage, SQL, 748

K

Klassen angeben, DEFINE CLASS-Statement, 153
Kleinbuchstaben, Umsetzung in Großbuchstaben, EXAMINE TRANSLATE-Statement, 290
Kommando-Prozessor, PROCESS COMMAND-Statement, 479

Komprimieren, COMPRESS-Statement, 130
 Konstante, definieren, DEFINE DATA-Statement, 169
 Koppeln von Dateien, FIND-Statement, 317

L

Länge einer Variablen verringern, REDUCE-Statement, 536
 Leerzeilen, generieren, SKIP-Statement, 611
 Lesen
 von Datensätzen
 FIND-Statement, 299
 GET SAME-Statement, 348
 GET-Statement, 344
 READ-Statement, 511
 SELECT-Statement, 735
 von Deskriptorwerten, HISTOGRAM-Statement, 355
 von Transaktionsdaten, GET TRANSACTION DATA-Statement, 352
 Level, in Datenstruktur, DEFINE DATA-Statement, 159, 177
 Limit, für Verarbeitungsschleife, LIMIT-Statement, 409
 LIMIT-Statement, 409
 Local Data Area. *Siehe* Data Area
 Löschen, Datensatz
 DELETE-Statement, 239
 DELETE-Statement (SQL), 722
 Logische Bedingung, IF-Statement, 362
 Lokaler Datenbereich. *Siehe* Data Area
 Loop. *Siehe* Verarbeitungsschleife
 LOOP-Statement, 412

M

Map
 Siehe auch Ausgabe
 aufrufen
 INPUT-Statement, 374
 WRITE-Statement, 656
 MARK-Option
 INPUT-Statement, 383
 REINPUT-Statement, 543
 Maske. *Siehe* Editiermaske; Map
 Mathematische Berechnungen. *Siehe* Arithmetik

Natural Statements Handbuch

Methoden, METHOD-Statement, 414
METHOD-Statement, 414
MOVE ALL-Statement, 432
MOVE-Statement, 419
Multiples Feld. *Siehe* Array
Multiplikation
 Siehe auch Arithmetik
 COMPUTE-Statement, 137
 MULTIPLY-Statement, 435
MULTIPLY-Statement, 435

N

Natural, -Session, beenden, TERMINATE-Statement, 641
NEWPAGE-Statement, 438
NOTITLE-Statement, 443

O

Objekt. *Siehe* Programm
OBTAIN-Statement, 444
ON ERROR-Statement, 445
OPEN CONVERSATION-Statement, 448
OPEN DIALOG-Statement, 449
OpenVMS, Begriffsdefinition, 2
Operanden, 5
Operandentabelle, 5
OPTIONS-Statement, 453

P

Parameter. *Siehe* Session-Parameter
Parameter Data Area. *Siehe* Data Area
PARAMETER-HANDLE-Definition, 164
Parameter-Datenbereich. *Siehe* Data Area
Paßwort, PASSW-Statement, 454
PASSW-Statement, 454
PA-Tasten. *Siehe* Funktionstasten
PERFORM BREAK PROCESSING-Statement, 466
PERFORM-Statement, 457

Periodengruppe. *Siehe* Array
 PF-Tasten. *Siehe* Funktionstasten
 POSITION-Option
 INPUT-Statement, 383
 REINPUT-Statement, 544
 Potenzierung
 Siehe auch Arithmetik
 COMPUTE-Statement, 137
 PRINT-Statement, 468
 PROCESS COMMAND-Statement, 479
 PROCESS GUI-Statement, 501
 PROCESS REPORTER-Statement, 503
 PROCESS SQL-Statement, 727
 PROCESS-Statement, 477
 Programm
 aufrufen
 FETCH-Statement, 294
 Nicht-Natural-Programm
 CALL FILE-Statement, 84
 CALL LOOP-Statement, 87
 CALL-Statement, 58
 ausführen, RUN-Statement, 573
 Ausführung abbrechen, STOP-Statement, 626
 -ende, END-Statement, 272
 Programmierung
 ereignisgesteuerte, 14
 komponentenbasierte, 14
 PROPERTY-Statement, 510

Q

Qualifizierung, von Datenstrukturen, 180

R

READ RESULT SET-Statement, 731
 READ WORK FILE-Statement, 526
 READ-Statement, 511
 Rechnen. *Siehe* Arithmetik
 REDEFINE-Option, DEFINE DATA-Statement, 167
 REDEFINE-Statement, 533

Natural Statements Handbuch

Redefinieren eines Feldes
 DEFINE DATA-Statement, 167, 177
 REDEFINE-Statement, 533
REDUCE-Statement, 536
 Länge einer Variablen verringern, 536
REINPUT-Statement, 537
REJECT-Statement, 16
RELEASE-Statement, 548
REPEAT-Statement, 551
Reporter, PROCESS REPORTER-Statement, 503
REQUEST DOCUMENT-Statement, 555
RESET-Statement, 568
RETAIN. *Siehe* FIND-Statement
RETRY-Statement, 571
ROLLBACK-Statement, 733
RPC
 CLOSE CONVERSATION-Statement, 98
 Kontextvariable, DEFINE DATA CONTEXT-Statement, 179
 OPEN CONVERSATION-Statement, 448
RUN-Statement, 573

S

Schirm. *Siehe* Map
Schleife. *Siehe* Verarbeitungsschleife
Seite
 Fußzeile, WRITE TRAILER-Statement, 665
 Seitenvorschub
 EJECT-Statement, 268
 NEWPAGE-Statement, 438
 Überschrift
 DISPLAY-Statement, 244
 NEWPAGE WITH TITLE-Statement, 438
 PRINT-Statement, 470
 WRITE TITLE-Statement, 661
 Verarbeitung am Anfang, AT TOP OF PAGE-Statement, 47
 Verarbeitung am Ende, AT END OF PAGE-Statement, 37
SELECT-Statement, 735
SEND EVENT-Statement, 576
SEND METHOD-Statement, 579
SEPARATE-Statement, 585
Session, abbrechen, TERMINATE-Statement, 641

- Session-Parameter, setzen
 - FORMAT-Statement, 340
 - SET GLOBALS-Statement, 595
- SET CONTROL-Statement, 594
- SET GLOBALS-Statement, 595
- SET KEY-Statement, 596
- SET WINDOW-Statement, 610
- SETTIME-Statement, 609
- Sichern. *Siehe* Speichern
- SKIP-Statement, 611
- Soft Coupling von Dateien, FIND-Statement, 317
- Sortieren von Datensätzen
 - FIND-Statement, 321
 - SORT-Statement, 613
- SORT-Statement, 613
- Sourcecode, dynamische Generierung, RUN-Statement, 574
- Spaltenüberschrift, DISPLAY-Statement, 245
- Speichern, Datensatz
 - INSERT-Statement, 724
 - STORE-Statement, 628
- SQL
 - alte Statements, 675
 - Common Set, 678
 - Datenbanken, Begriffsdefinition, 2
 - Extended Set, 678
 - flexible, 712
 - Statements, 675
- Stack
 - Daten ablegen, STACK-Statement, 621
 - löschen, RELEASE-Statement, 548
- STACK-Statement, 621
- Start-of-Data-Verarbeitung, AT START OF DATA-Statement, 43
- Statements, 7
 - bedingte Verarbeitung, IF-Statement, 362
 - SQL, alte, 675
 - SQL-Statements, 675
 - Überblick, 7
- STOP-Statement, 626
- Stored Procedure, CALLDBPROC-Statement, 716
- STORE-Statement, 628
- Struktur, von Operanden, 5
- Stufe. *Siehe* Level

Natural Statements Handbuch

Subprogramm, ausführen, CALLNAT-Statement, 89
Subroutine
 aufrufen, PERFORM-Statement, 457
 definieren, DEFINE SUBROUTINE-Statement, 208
SUBSTRING-Option
 EXAMINE TRANSLATE-Statement, 291
 EXAMINE-Statement, 283
 MOVE-Statement, 423
 SEPARATE-Statement, 586
SUBTRACT-Statement, 635
Subtraktion
 Siehe auch Arithmetik
 COMPUTE-Statement, 137
 SUBTRACT-Statement, 635
SUSPEND IDENTICAL SUPPRESS-Statement, 638
Syntaxsymbole, 3

T

Tabelle. *Siehe* Array
Tasten. *Siehe* Funktionstasten
Teilung. *Siehe* Division
Terminal, -kommandos, im Programm setzen, SET CONTROL-Statement, 594
TERMINATE-Statement, 641
Textformatierung, COMPOSE-Statement, 105
Textvariable, SQL, 714
Titel
 DISPLAY-Statement, 244
 NEWPAGE WITH TITLE-Statement, 438
 PRINT-Statement, 470
 WRITE TITLE-Statement, 661
Top-of-Page-Verarbeitung, AT TOP OF PAGE-Statement, 47
Transaktion
 abbrechen
 BACKOUT TRANSACTION-Statement, 51
 ROLLBACK-Statement, 733
 Ende
 COMMIT-Statement, 721
 END TRANSACTION-Statement, 273
 Transaktionsdaten lesen, GET TRANSACTION DATA-Statement, 352

U

Überschrift, unterdrückt, NOTITLE-Statement, 443
 Überschrift. *Siehe* Seitenüberschrift; Spaltenüberschrift
 Übertragen von Daten in ein anderes Feld
 COMPUTE-Statement, 137
 MOVE ALL-Statement, 432
 MOVE-Statement, 419
 Uhrzeit. *Siehe* Zeit
 UNIX, Begriffsdefinition, 2
 Unterbrechen. *Siehe* Abbrechen
 Unterdrückte Überschrift, NOTITLE-Statement, 443
 Untersuchen eines Feldwertes, EXAMINE-Statement, 282
 UPDATE-Statement, 643
 UPDATE-Statement (SQL), 749
 UPLOAD-Statement, 648

V

Variablen. *Siehe* Benutzervariablen; Globale Variablen
 Verarbeitung, bedingte. *Siehe* Bedingte Verarbeitung
 Verarbeitungsschleife
 Anzahl der Durchläufe, begrenzen, LIMIT-Statement, 409
 initiiieren, FOR-Statement, 337
 schließen, LOOP-Statement, 412
 verlassen, ESCAPE-Statement, 278
 wiederholt ausführen, REPEAT-Statement, 551
 View, DEFINE DATA-Statement, 156, 166

W

Wert, Ausgangswert, für Array, DEFINE DATA-Statement, 175
 Window
 DEFINE WINDOW-Statement, 214
 INPUT WINDOW-Statement, 378, 393
 SET WINDOW-Statement, 610
 Windows, Begriffsdefinition, 2
 Work File. *Siehe* Arbeitsdatei
 WRITE TITLE-Statement, 661
 WRITE TRAILER-Statement, 665
 WRITE WORK FILE-Statement, 670

Natural Statements Handbuch

WRITE-Statement, 649

Z

Zeichen, Umsetzung von, EXAMINE TRANSLATE-Statement, 290

Zeit, setzen, SETTIME-Statement, 609

